NASA / GODDARD SPACE FLIGHT CENTER

Advanced Architectures and Automation Branch

NGST Scientist's Expert Assistant Final Report

April 2000

Chris Burkhardt
Mark Fishman
Sandy Grosvenor
Jeremy Jones
Anuradha Koratkar
LaMont Ruley
Karl Wolf

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland

NGST Scientist's Expert Assistant (SEA) Final Report

April 2000

Prepared by:

Jeremy E. Jones Date
Advanced Architectures and Automation Branch

Approved by:

Julie Breed Date
Advanced Architectures and Automation Branch

Approved by:

Keith Kalinowski Date
NGST

Goddard Space Flight Center
Greenbelt, Maryland

TABLE OF CONTENTS

1 Preface

1.1 Purpose

This document contains the final report of the Next Generation Space Telescope (NGST) Scientist's Expert Assistant (SEA) project. It describes lessons learned during the course of the project and results from the evaluation of the software. This document was written for the NGST project and the Advanced Architectures and Automation branch of Goddard Space Flight Center's Information Systems Center (ISC).

1.2 Audience

This document is targeted at three overlapping audiences: (1) managers involved in NGST development, who we anticipate are knowledgeable of science and astronomical concepts, and aware of software technology concepts, but not experts; (2) more technical software engineers interested in the concepts we've tried and our successes and failures; (3) astronomical readers who are involved in observatory operations in general. The technology portions of this document are written with the assumption that the reader is generally familiar with the technology being discussed, including Java and XML. Other portions of this document also assume a basic understanding of astronomical concepts. Throughout, , we have tried to describe our work in terms that will not overwhelm readers that are not experts in both astronomy and computer science

1.3 Document Organization

This document is organized into several sections. An executive summary begins the document. Section 3 is a more detailed introduction to the SEA and its capabilities. Section 4 describes the philosophy behind user involvement in the SEA. Section 5 details the technological challenges and lessons learned during the project. Section 6 describes the process behind, and the results of, the evaluation phase. Section 7 describes the challenges and lessons learned in collaborating with other groups. Section 8 details possibilities for future development both within the SEA and as spin-off efforts. Finally, section 9 provides highlights and conclusions from the SEA effort.

1.4 Acknowledgements

This work was funded by the Next Generation Space Telescope technology funds. The SEA is a joint effort between the Space Telescope Science Institute (STScI) and the Advanced Architectures and Automation Branch of Goddard Space Flight Center's (GSFC) Information Systems Center. We also thank our evaluators, the staff at the Space Telescope Science Institute, Gemini, ESA, and ESO for their invaluable insight and recommendations.

2 Executive Summary

The Scientist's Expert Assistant (SEA) has been an effort to develop innovative, software-based tools to better enable scientists to develop valid observational proposals for using the Hubble Space Telescope (HST). In order to meet the operational cost objectives for the Next Generation Space Telescope (NGST), this proposal preparation process needs to be dramatically less time consuming and less costly. The goals and philosophies used in developing SEA have attempted to utilize an interactive visual and expert system approach to make the user more self-sufficient and hence minimize manual effort and cost for user support.

The SEA team worked with the user community through the addition of a practicing astronomer to the core team. While primarily acting as an advocate of end-users and sharing the astronomical expertise of our user community, this team member also knew (or learned) enough of the technical design language to provide an invaluable link between the technical computer developers and the end-user community. A formal evaluation phase was also extremely successful in connecting with the user community, so much so that we strongly recommend incorporating formal end-user evaluations throughout the development and testing cycle of a new product.

The SEA team has been aggressive in applying new technology to the SEA design. The SEA is written in Java, which enables cross-platform deployment and provides a wealth of capabilities. XML (Extensible Markup Language) is used for proposal storage and for configuration and preference files. The SEA team also tested the viability of using expert systems to assist the observer in planning their proposals. Expert system use in the SEA has evolved over the course of the effort, and includes a proactive interface that guides the user through choosing an instrument, detector and filter combination, plus a dither module that takes a more passive approach, allowing the user to work with an interactive user interface, while the expert system monitors the user's choices and recommends actions based on those choices. While the original scope of expert system use within the SEA was never fully realized, we still believe that expert systems hold great promise for future software systems.

During February 2000, we conducted user evaluations on SEA. Our evaluators were chosen from astronomers with accepted HST Cycle 9 programs that focused on HST’s WFPC2. This evaluation was to determine if we had succeeded in achieving our objectives of having an intelligent, intuitive, distributed, adaptable, integrated, and flexible system. We were also interested to learn how usable the SEA system was. On a scale of 1 - 5, where 1 was excellent and 5 was poor, the SEA as a whole ranked as 1.8 (i.e., between excellent and above average). All our evaluators (except one!) found using SEA a better and easier experience than RPS2 (the present tool used at STScI to develop proposals). In terms of creating accurate and feasible observations we feel that the SEA provided the proof of concept.

In addition to developing the SEA as a test-bed for new visual approaches, we’ve learned that the really substantial reductions in the costs of general observer programs will come not from simply applying technology, but from establishing a common core of observing tools across all major observatories, ground or space based, inside or outside of NASA. Having a common core will allow all observatories to save the costs of proprietary development, testing, and maintenance. It will also improve science by saving the end-user astronomers the costs of learning multiple tools, allowing them to gain significant comfort and expertise in a single suite of tools applicable to many observatories. When the expertise of the end-user in a tool suite grows, support demands on the observing staff should decrease and the quality of the proposals and subsequent science should increase. Into the future, a common core will continue to save as observatories can work together to evolve and improve their software tools, putting software R&D efforts into research, that no single observatory can afford to do on their own, and thus keeping the software tools from stagnating until they become an operations crisis.

3 Introduction

In the summer of 1997, the Operability Team for the Next Generation Space Telescope (NGST) approached Code 588 about researching ways to substantially reduce the level of effort involved in supporting a General Observer program for an observing mission. This document is the final report from that project.

During this time, a small group of astronomers and computer scientists form the Space Telescope Science Institute (STScI) and Goddard's Advanced Architectures and Automation (Code 588) group has studied and researched several alternatives and have developed a functional prototype. That prototype, the Scientist's Expert Assistant (SEA), uses an interactive visual and expert system approach to developing and planning observing programs, i.e. the proposal preparation process.

In addition to developing the SEA as a test-bed for new visual approaches, we've learned that the really substantial reductions in the costs of general observer programs will come not from simply applying technology, but from establishing a common core of observing tools across all major observatories, ground or space based, inside or outside of NASA. Having a common core will allow all observatories to save the costs of proprietary development, testing, and maintenance. It will also improve science by saving the end-user astronomers the costs of learning multiple tools, allowing them to gain significant comfort and expertise in a single suite of tools applicable to many observatories. When the expertise of the end-user in a tool suite grows, support demands on the observing staff should decrease and the quality of the proposals and subsequent science should increase. Into the future, a common core will continue to save as observatories can work together to evolve and improve their software tools, putting software R&D efforts into research, that no single observatory can afford to do on their own, and thus keeping the software tools from stagnating until they become an operations crisis.

Observing, be it ground-based or space-based or classical or service mode, is a pipeline from start to finish. It starts with submission of a proposal and ends with properly documented data sets in the archive. In the present era, it is often the case that one group develops the observing instruments, while the instrument is commonly used by another group of people. Further, the instrumentation can be very complex with large capital investment both in instrumentation and observatory infrastructure. In such an environment, for any type of observing to maximize scientific returns, it becomes challenging when it has to be achieved with limited resources for user support. This goal can be achieved by understanding the observational conditions, instrument capabilities, and scheduling complexities thoroughly, and providing this information to the user community in a concise, timely, effective and efficient manner. Thus, an observatory staff depends on:

(a) Documentation
(b) Software tools
(c) Human user support

to disseminate information effectively to the user community. But in an era of limited resources, user support provided by a person has to be minimized without losing the human aspect of support. The SEA effort has based its philosophies and goals on determining how documentation, software tools, and human user support can be efficiently optimized using new technology.

3.1 Choice of Observatory

3.1.1 Why Hubble and STScI as a Baseline

Early in the project, we were faced with the need to establish a close relationship with an existing observatory to interview its staff and to provide a baseline for our research and software development. Since NGST was still in an early stage, strategies on how to provide user support, observational modes etc.

had not yet been developed. Since NGST will be a space-based observatory operated in service mode (i.e., where observers provide observing information to the observatory and the observations are scheduled according to the instructions obtained from the observer), we wanted to prototype an observatory whose existing operational format was similar to the likely format for NGST. Since our team is based at Goddard Space Flight Center in Greenbelt, a nearby facility was also important. The Hubble Space Telescope (HST) and STScI were obvious choices as HST is operated 100% in service mode and STScI has been providing HST's user support for the last decade. Further, STScI will also be the operations center for NGST.

While HST has provided our baseline, we believed early on that substantial productivity gains could be made by having a modular, multi-observatory system, and have taken care to make the prototype's architecture as modular as possible.

3.1.2 Proposal Preparation Process at STScI

In service mode operations, the proposal preparation process is normally divided into two phases:

- The Phase I: In this phase the proposer describes a project, its scientific goals and objectives and defends the proposed science to a group of his/her peers. A proposal in this stage is the basis for the initial planning and scheduling by the observer and observatory staff.
- The Phase II: In this phase the successful proposer provides the observatory staff with a precise executable set of observations to be made

In order to develop an effective program, during the proposal preparation process observers need to understand many aspects of the observatory's operations, such as: how the observatory schedules its observations; what instrumentation is available at the observatory; what are the characteristics of the observatory/instrumentation; and how much observing time will be required for a given science project. The observatory staff must provide all the above information to the users.

It is crucial to have an effective proposal preparation process. Problems in the beginning of the observing pipeline have a tendency to propagate down the pipeline and make the entire system costly and inefficient. For large observatories, the proposal preparation process must result in observers successfully submitting well-defined, accurate, unambiguous, flexible, feasible and schedulable programs. As a secondary objective, the proposal preparation and scheduling processes should also be conducive to information dissemination.

For these reasons, the SEA project has concentrated on the proposal preparation process, i.e. the Phase I and Phase II process from the user's perspective. Note that we will not be differentiating between the Phase I and Phase II because a good proposal/observing program uses many of the same tools during both phases although the style in which the tool is used may be different in the two stages. For example, in Phase I the exposure time calculator (ETC) is used as an exploratory tool, while in Phase II the same ETC is used to make small changes in the Phase II program and to determine the effect of these changes on the scientific objectives of the observing program. We also discovered during testing SEA a new "phase" which we nicknamed Phase 0. During Phase 0, a good visual tool such as SEA is used as an exploratory, research tool to examine the initial feasibility of an idea, even before the idea has been refined into a specific Phase I proposal.

3.1.3 Available Tools

STScI has invested heavily in tools, as well as expert human support that can be used during proposal preparation. These tools have evolved over time in an attempt to keep up with both technology and observatory operations strategy. This approach has been used successfully throughout the HST mission and at other observatories as well. Originally, Hubble proposals were submitted using the Remote Proposal Submission System (RPSS). This submission system was designed in the mid-1980s and represented the state of technology and experience in handling "service mode" observations. It provided the bare minimum of user support, for example, checking for syntax or spelling errors and some illegal configurations. When received at STScI, these proposals were processed to determine feasibility and schedulability. It was at this stage that most problems were discovered, and manual intervention by operations staff was necessary. RPSS was used until 1994. RPSS was neither conducive to efficient user support strategies nor was it observer/observatory staff user friendly.

After three years of proposal preparation experience, an effort was initiated to improve upon the RPSS process, which led to the release of RPS2 for cycle 5 (1994) observing. RPS2 was designed to further two major goals:

- Improve the quality of proposals at submission time and thereby avoid the need for observers and operations staff to iterate. This was achieved by making some of the telescope operations constraints available to observers when they prepared their programs.
- Make routine the process of updating proposals after submission for scientific or operational reasons. This was achieved by dividing proposals into "visits" which can be independently planned and scheduled.

RPS2 was implemented using client/server technology that processed a proposal in batch and then displayed the results of the processing to the observer. In developing RPS2, STScI opted in favor of the modest RPS2 architecture instead of a full interactive environment because of the shortcomings of the existing scheduling system and the lack of suitable integration software like CORBA and Java RMI which were still years away. Further, the simpler RPS2 architecture, which was a vast improvement over RPSS, could be provided to the user community quite rapidly. A fully interactive system was thus not cost effective in 1994.

3.1.4 Potential for Improvement

Upon analyzing the user support strategies at STScI, we determined major areas where user support (both software tool and expert human support) can be improved so that astronomers do not encounter barriers as they prepare their proposals.

- The current approach for software development builds tools with artificial boundaries. Astronomers are given a variety of largely independent tools (for example, in the case of HST, Exposure Time Calculators, Phase I Template, RPS2, StarView, STSDAS) that do not communicate well (if at all) with each other, and *often have significant learning curves*. In addition, the astronomer must be familiar with hundreds of pages of documentation (Call for Proposals, Phase II Instructions, Instrument Handbooks, Data Handbook, WWW pages, etc.). It is presently very difficult for users or observatory staff to efficiently find information contained in the documentation. Further, software tools can provide very little assistance to answer users' questions since documentation is largely unstructured text. As a consequence of this complexity, at the STScI, Program Coordinators and Contact Scientists (staff in charge of user support) must review every observation, provide extensive user support, and write a large amount of documentation. This directly increases the workload of STScI staff.

- The current suite of tools are also very text or form oriented. Few tools were available at the beginning of the SEA project to provide astronomers with a visual approach to defining their science. The lack of visual tools has been partly due to lack of power at the desktop level, and partly due to lack of software development dollars and effort. With SEA we had the ability to perform the research and development while the continuing growth in desktop power and new tools such as Java made the visual approach a possibility.

- Expert human support is currently focused on routine activities. Direct staff support is currently a manually intensive (and therefore costly) process, whether it is direct interaction with observers or writing documentation, but the goal of human-to-human interaction should be the innovative and extraordinary, not the routine. It is also currently cumbersome to capture expert knowledge into a form that is easily used by humans and software. The traditional way of capturing this information is by writing software tools, but this is an expensive and slow process. The computer science disciplines of artificial intelligence and expert systems are continuing to evolve in both their capability to process "expert knowledge" and in the tools to enable capture of the experts' knowledge. We felt that experimenting with expert systems was important. If expert systems tools could handle the current manual effort of supporting the more repetitive, routine activities, then the observatory staff would be free to focus on the truly innovative and extraordinary proposals.

3.1.5 Selection of Testbed

To determine the effectiveness of the SEA ideas we wanted to establish a testbed instrument so that we could have a test platform for SEA while providing a way to compare SEA to existing tools. We initially worked with HST's Advanced Camera for Surveys (ACS) because ACS provided us with a real instrument with a good balance of complexity and operational style when compared to the expectations for NGST. At the time, ACS was scheduled for installation such that existing tools were being adapted to handle it. We expected to be able to compare operational ACS cases against our SEA test-bed. However, the ACS launch was delayed and evaluation of SEA needed to be completed by March 2000. Consequently, we changed our testbed instrument to HST's Wide Field Planetary Camera 2 (WFPC2).

Changing of the testbed mid-stream provided us with an opportunity to see if ideas developed for one instrument could be easily extended to a new instrument. We found it relatively easy to do this transition. Yet, there were a few "gotchas". Some features of the two instruments, while scientifically similar to the end-user, were implemented very differently for the two instruments. An example of this is the CR-SPLIT parameter, which is the splitting of a single lengthy exposure into two or more shorter exposures to counteract the occurrence of cosmic rays. It is implemented in different ways for the ACS and WFPC2. On ACS, the user specifies an integral number of "sub-exposures" and the overall exposure time is automatically divided between sub-exposures. On WFPC2, the CR-SPLIT parameter is a Boolean flag and when the user specifies yes, the exposure is automatically split into two sub-exposures. It is a relatively small problem for the software to overcome, but it illustrates the need for an instrument expert to be involved in the development of the tools from the beginning. By involving both instrument experts and user-experts early, commonalties between instruments and operation modes can be seen up front during the design phase of the tools.

3.2 Philosophies and Goals

"Good" proposal preparation software tools help decrease the amount of human user support, and provide an easier interface to understanding the complexity of both instruments and observing programs. They allow users to explore their parameter space. We define a "good" software tool as a tool that has the following properties:

- Easy to use for users of varying degrees of expertise
- Provides reduction in manual support from observatory staff. This implies that the tool decreases complexity of instruments/processes, provides easy access to up-to-date reference information, is flexible and helps guide the user towards completion of that tool's process.
- Allows exploration of the observing parameter space
- Allows visualization and is as interactive as possible
- Allows documentation to be an integral part of software tools
- Is aesthetically pleasing
- Is common for both observatory staff and observer

The main philosophies for development of the SEA were:

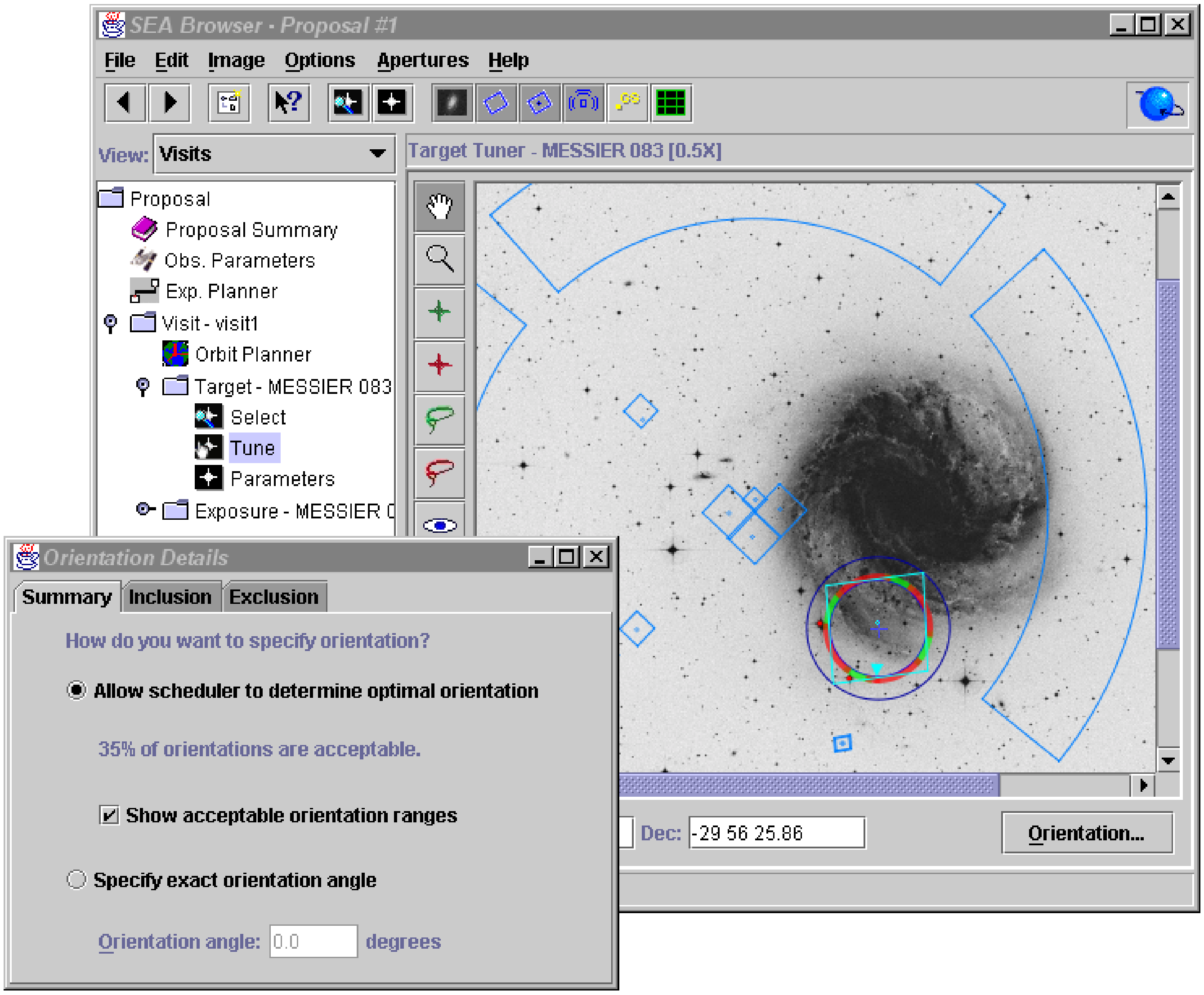
- The system should be *intelligent*. It should employ artificial intelligence methodologies and paradigms to assist and guide the scientist in producing a proposal that is flight ready.

- The system should be *intuitive*. The user interface should not require extensive training. Scientists should be able to use the SEA with little or no assistance.
- The system should be *distributed*. It should allow delivery and processing of proposals via the World Wide Web across a wide range of computing systems.
- The system should be *adaptable*. As the telescope staff learns how best to use NGST once it is launched and operational, the system should be able to incorporate new information and knowledge easily.
- The system should be *easily integrated* with other NGST planning and operations modules.
- The system should be *flexible*. Since NGST is not scheduled for almost a decade, the system development must allow for changes in technology. Further, much of the process of developing observing proposals is common among observing platforms. This system could and should be an effective alternative for other observatories, both present and future.

3.3 System Overview

In SEA, users can retrieve and display previously observed images of astronomical targets, overlay any of HST's instrument apertures on the image. Further, by "clicking and dragging" they can visually define some of the parameters of their observations. For example, the probability of scheduling can be significantly affected by specifying particular aperture orientations. In SEA, users can interactively rotate the aperture and see the impact of their orientation constraints (see Figure 1). A built-in Exposure Time Calculator allows users to graphically display the impact of changes in target properties and instrument setups on exposure time and/or signal-to-noise (see Figure 2). The SEA includes an Orbit Planner for visualizing the orbits within a visit and specifying constraints between exposures (see Figure 3) plus a Visit Planner for graphically specifying constraints between visits. The SEA also includes editable table views (see Figure 4) of all information that allow the user to see all observation parameters at a glance and manipulate them in one display. In addition, a rudimentary expert assistant can quickly guide new users through the various detector/filter combinations and recommend a combination that fits the users' scientific needs. SEA has also taken the first steps to integrate documentation with software, providing preliminary context-sensitive help, and access to reference data.

Figure 1: SEA's Visual Target Tuner



How do you want to specify orientation?

(●) Allow scheduler to determine optimal orientation

35% of orientations are acceptable.

[✓] Show acceptable orientation ranges

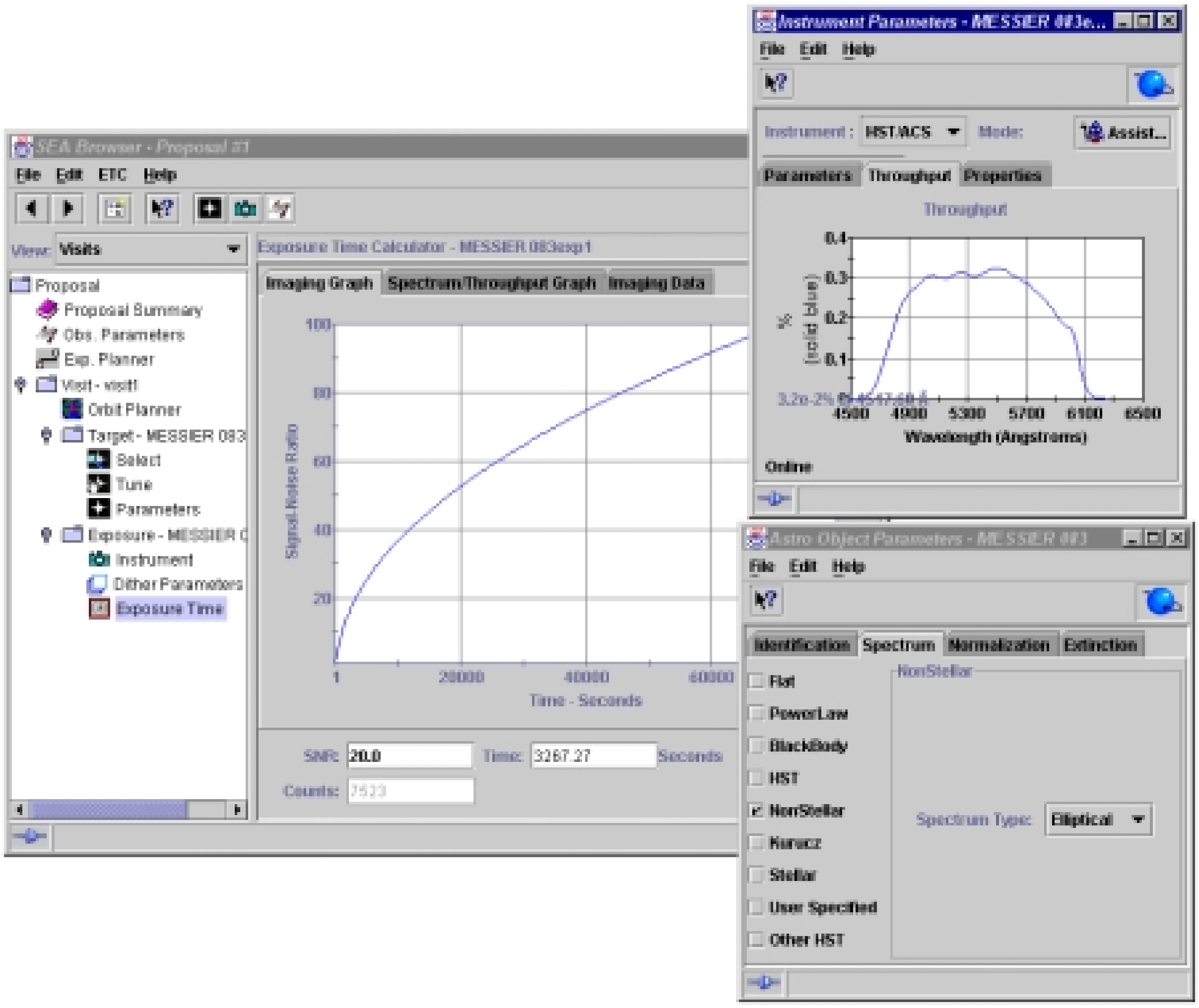
( ) Specify exact orientation angle

Orientation angle: 0.0 degrees

SEA is based on a component architecture that shares a common underlying object-oriented design and allows the major components to function either as independent modules, or as an integrated whole under the "Proposal Browser." Current modules include a Visual Target Tuner, an Exposure Time Calculator, a Visit Planner, an Orbit Planner, and the integrating Proposal Browser component. By developing the system using the Java programming language, we have a true multi-platform system that operates on Microsoft Windows, Solaris, Linux and other Unix platforms with no modifications.

The technical approach to developing SEA has been an iterative rapid prototyping approach. Informally known as "design-a-little-build-a-little-test-a-little," this approach involves iterating through the design/build/test cycle allowing new ideas to be quickly explored, and if promising to be developed further. If a particular feature does not show effectiveness, it can be abandoned before excessive resources have been invested.

Figure 2: SEA's Exposure Time Calculator



View: Visits

Instrument: HST/ACS Mode:

SNR: 20.0 Time: 3267.27 Seconds

Counts: 7533

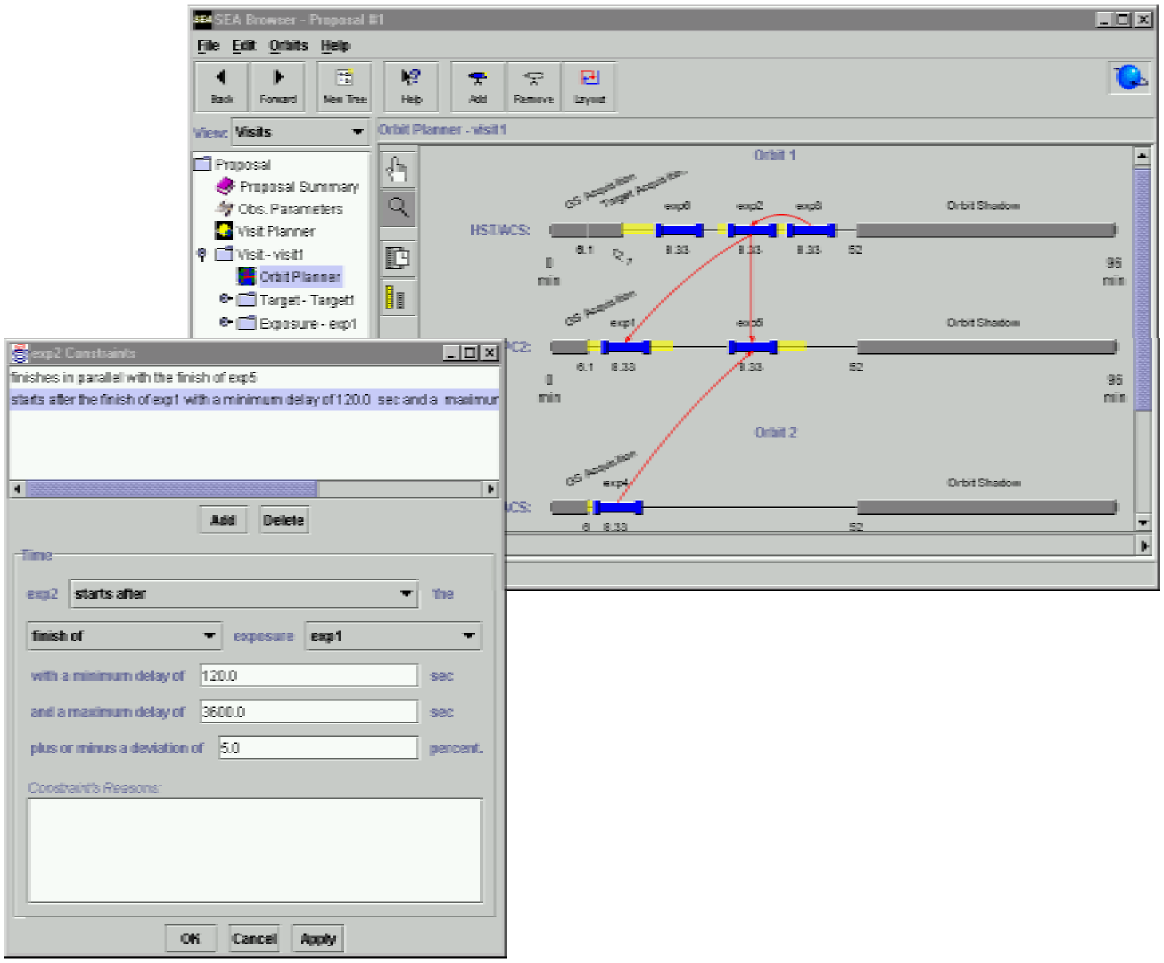
[✓ NonStellar]

Spectrum Type: Elliptical

4 User Involvement and Team Interaction

SEA's approach to incorporating user input has been an iterative approach that has incorporated several different components. As one of the first steps to the project, we conducted interviews with several astronomers, and STScI staff about their perceptions of the strengths and weaknesses of their current tools. This included some initial brainstorming about what types of new tools might have the most significant impact. Throughout the project life, we have tried to demonstrate SEA at major astronomical conferences (see Appendix B). The feedback we've obtained through these conferences has had a significant impact on the project. Lastly, we recently ran the SEA through a more formal evaluation phase.

Perhaps the most critical component of incorporating user involvement has been the addition of a practicing astronomer as a full member of the team. We have coined a new term in computer usability for such a team member: "alpha-user." While most of the team is composed of computer scientists with little formal astronomical training, our "alpha-user" has several years of experience at STScI in user-support with little formal computer training. An alpha-user acts not only as a primary direct source of end-user input, but also coordinates input from many other users. While primarily acting as an advocate of end-users and sharing



finishes in parallel with the finish of exp5
starts after the finish of exp1 with a minimum delay of 120.0 sec and a maximur

Time

exp2 starts after the
finish of exposure exp1
with a minimum delay of 120.0 sec
and a maximum delay of 3600.0 sec
plus or minus a deviation of 5.0 percent.

Constraint's Reasons:

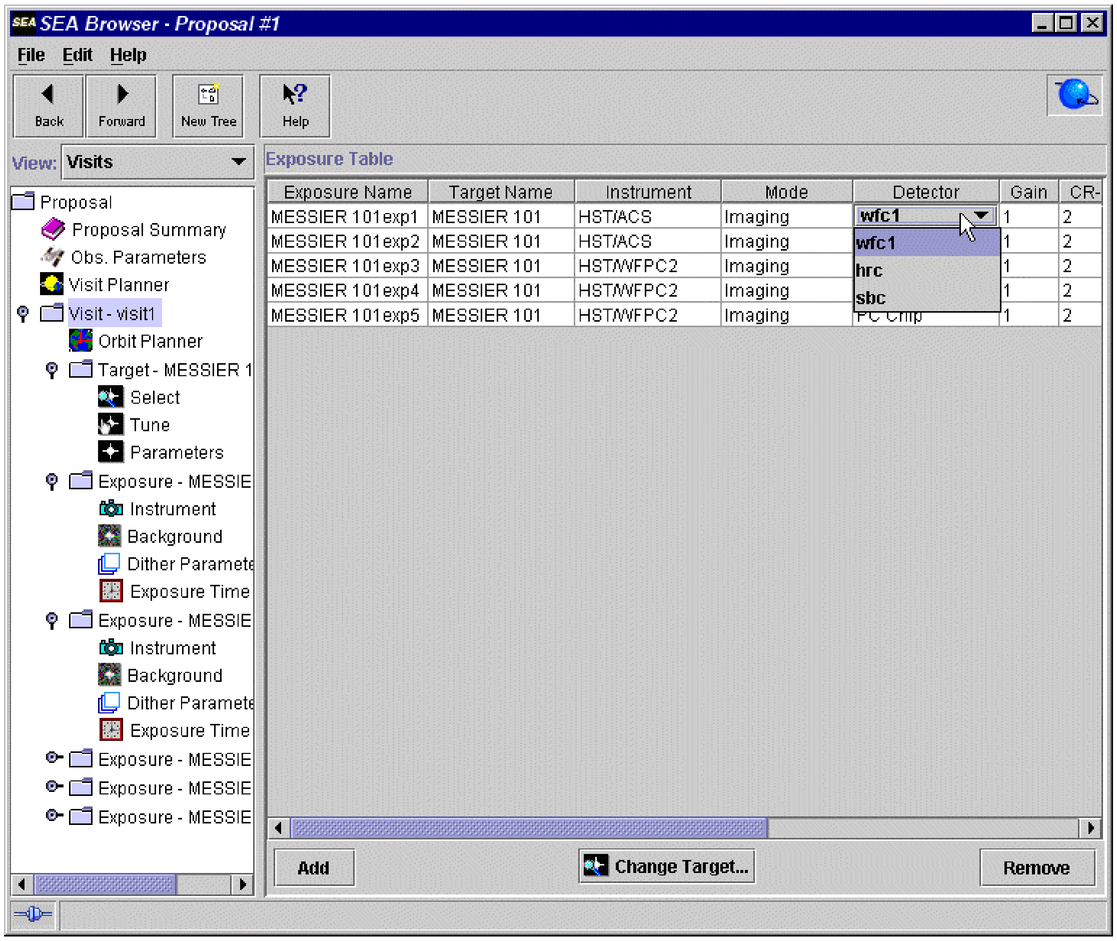
Figure 3: SEA's Orbit Planner

the astronomical expertise of our user community, an alpha-user also knows (or learns) enough of the technical design language to provide an invaluable link between the technical computer developers and the end-user community. SEA has benefited hugely from this experience. The team needed a strong end-user with expertise in astronomy as well as a vision of the types of tools needed.

During February 2000, after two years of informal demonstrations, SEA underwent a more formal evaluation period. We established a list of specific tasks for our evaluation team of working astronomers to perform. We then observed them as they performed those tasks providing minimum coaching necessary. Details on the results of this evaluation phase can be found in section 6. This phase has been extremely beneficial. Having a set of novice SEA users perform a set of specific tasks added a whole new level of feedback. We flushed out a number of undiscovered bugs, got some excellent ideas, and learned a tremendous amount about what was and was not intuitive. We strongly recommend incorporating formal end-user evaluations throughout the development and testing cycle of a new product. User evaluations (early and often) throughout the development cycle will save valuable time and effort.

An often-asked question about software development is: Can "rapid prototyping" really work? For SEA the answer has been a resounding yes. Our objective during SEA's development has been to test and evaluate new visual concepts, not generate a production system. However, we also knew at the beginning that even as a prototype, the software engineering needed to be sound, or it would not support the infrastructure that we knew SEA had to support. One of the significant aspects of SEA has been a very

Figure 4: SEA's Table Views



Exposure Table

Exposure Name	Target Name	Instrument	Mode	Detector	Gain	CR-
MESSIER 101exp1	MESSIER 101	HST/ACS	Imaging	wfc1	1	2
MESSIER 101exp2	MESSIER 101	HST/ACS	Imaging	wfc1	1	2
MESSIER 101exp3	MESSIER 101	HST/WFPC2	Imaging	hrc	1	2
MESSIER 101exp4	MESSIER 101	HST/WFPC2	Imaging	sbc	1	2
MESSIER 101exp5	MESSIER 101	HST/WFPC2	Imaging	PC Chip	1	2

loose coupling of requirements to system features. Since we were intentionally exploring new interface concepts, we had very few specific requirements. The rapid prototyping allowed us to adjust to an evolving and ever-changing set of requirements and priorities. That coupled with the object-oriented approach has also allowed the underlying architecture of SEA to evolve and grow with the system.

5 Technology

5.1 Java

At the inception of SEA, Java was just beginning to emerge as a truly viable development language. Its claims of cross platform independence, its extensive libraries, and its ability to run from the web all looked like a good fit for meeting SEA's goals. Therefore, the decision was made to design and implement SEA as a pure Java product. Since this decision there have been many issues and trials but overall the feeling of the team has been that this decision was the correct one. It is likely that Java will continue its trend towards becoming the predominant development platform, and NGST or any other mission should consider Java first for future software development.

One of the early-recognized strengths of Java was the availability of many built-in and add-on libraries. Libraries such as Java Advanced Imaging (JAI) and Java Project X (XML) as well as the more standard

Swing and Java2D libraries allowed us to focus our effort on developing higher level, product specific components. These built-in libraries made for a substantially smaller development time than might have been expected given a less robust set of development tools and it is our belief that SEA would not be nearly as rich a tool without the advantages that these libraries provided.

On the other side of our decision to use Java, one of the early areas of concern for SEA was performance. Java, with its use of byte code and a virtual machine (VM), has frequently been blamed for performance problems. Performance has at times been an issue for SEA. We adapted partly through some design modifications, but more often Sun resolved the issues in subsequent Java releases. The optimizations we made included everything from redesigning algorithms to just placing large tasks on a separate thread to allow the user to continue while the task completed. Probably one of our most notable performance problems dealt with the startup of the application. The starting of the VM, the loading of all JARs (Java Archives), and the initialization of the SEA application could frequently take 20 seconds or more. To help minimize this cost we threaded what we could of this startup and made sure that a splash screen came up at the earliest moment possible. This solution, while not perfect, seems to be acceptable. However, there is hope for the future. Based on documentation from Sun as well as some preliminary, fairly unscientific performance tests the Java 1.3 client VM significantly improves Java's performance. For example, the startup of SEA went from 20 seconds to 11 seconds. Other areas of SEA also received performance improvements from the new VM.

Another issue that was less obvious but quickly became a concern for us was Java's rapid life cycle. The SEA project stretched from the beginnings of Java 1.1 until the current candidate release for Java 1.3. During this time Java went through several evolutions. Bugs were fixed. New bugs were created. Libraries changed. APIs changed. Even class behaviors changed in certain cases. However, surprisingly, the majority of these changes required relatively low amounts of rework in the SEA code. There were times where features needed to be disabled until a bug fix came out or occasionally method parameter lists changed but in general it was our experience that Sun went out of its way to make the impacts of their changes as small as possible.

One important area for the SEA team dealt with Java's cross platform abilities. The "write-once-run anywhere" claim was a key feature in choosing Java as our development language. We needed to support customers who worked in Unix, NT, and Apple environments. While there have been issues relating to this cross platform capability, some minor problems with fonts and layouts, all core functionality ended up porting without modification. Further, many of the problems with cross-platform consistency have been eliminated by Sun as Java has evolved. We developed SEA on NT but were able to deploy it on Solaris, Linux, and even DEC Alpha machines. At the time of this paper, Apple still has not released a version of Java 1.2 for the Macintosh, so SEA was unable to run in that environment but other than that we were pleased with Java's support of this capability.

One of our early goals called for SEA to run as both an application and an applet. While not absolutely necessary this requirement appeared to be fairly easy to implement and was useful in allowing users to always run the latest version of the application. However, we concluded that applets are best for simple applications with no major security requirements. The reason for this decision dealt with three problems that arose during development:

- **Application Size**
  Between the SEA classes and third party libraries, we found that our distribution file ballooned to a multi-megabyte size (approximately 8 megs). This is substantially beyond the size threshold for reasonably running as an applet.

- **Security Model**
  In JDK 1.1 the security model was coarse and each browser vendor was expected to implement their own version of it. With the coming of the plug-in and JDK 1.2, Sun provided a more fine-grained security model that could be treated the same across all browsers. Unfortunately, while

the model itself was better, it required the end-users to partially manage the security setup on their own system. We felt that these end-user setup requirements were too complicated for the average user. While ideas such as developing an installer to setup the applet security were discussed, the decision was eventually made to just drop the applet support entirely.

- **Browser Support**
  The third problem with applets is the ongoing lag in major browsers supporting new versions of Java, and of incompatibilities between different browsers. While using Sun's Java Plug-in helped considerably, it required the end-user to install the plug-in, which eliminates much of the advantage of an applet's "transparency".

In dropping support for applets, we still needed a flexible distribution solution that was as simple as possible for our end-users. Eventually we decided on a two-prong method for distributing SEA and keeping it up to date. First, we decided to use InstallAnywhere by ZeroG Software to distribute SEA. InstallAnywhere is a third party tool that allowed us to create installers to distribute SEA, the Java runtime, and any third party libraries on both NT and Unix environments. Further, to make sure that SEA was up to date, we added the ability for it to check its version against the newest version of SEA on the SEA distribution site and download a copy if a newer version existed.

One of the major lessons that we learned while developing SEA was that by taking advantage of the built-in features of Java it is possible to come up with a very plug-able/extensible type of infrastructure. By using common object-oriented principles like interfaces and base objects in the SEA design, the SEA allows the extension of these objects without the need to modify the classes that use those objects. Additionally, by using common software design "patterns" coupled with Java features like dynamic class loading and introspection we have developed an architecture where features can be changed and new components can be added to SEA without the need to modify existing components. One of our better decisions was to model SEA after the Model/View/Controller (MVC) design pattern, which describes a way to separate the system's data from the various user interfaces to the data, while ensuring that changes to data are propagated properly throughout the application. While this pattern is not unique to Java, it is one that the Sun developers emphasize and we found that it made integration much easier. By using one model, which many views shared, we were able to have any changes made by one view seamlessly reflected in all of the others.

5.2 Expert Systems

Early in the development of the SEA, we recognized there was a great deal of knowledge, both astronomical and procedural, required by an observer to create an observation proposal. From the beginning, the SEA's primary goal was to make it easier to create observing proposals. We therefore felt it would make the development process easier if we could build this expert knowledge into the SEA. The intent was to aid the novice user while acting as a double-check for the experienced observer. However, it has not been easy to determine the best way to integrate the expert system technology into the SEA. We have made three attempts with varying degrees of success. The users need to intuitively know the simplicity or complexity of their scientific program, but procedural needs may not give them confidence in what they have learned. Thus, from our evaluation (see section 6) we have found that further research into expert systems needs to continue, because the users constantly indicated the difficulty of understanding the complexity of the new instruments and observing strategies.

5.2.1 A Brief overview of Expert System Technology

Expert System Technology is an applied branch of Artificial Intelligence research, which focuses on techniques to incorporate detailed domain expert knowledge into software systems. The knowledge is usually embodied in something called a "rulebase", which is a collection of rules, and the data used by those rules. The rules model the domain logic (aka "Business Logic") and usually have a simple IF-THEN-

ELSE format. They are intended to be readable (they say) by non-programmers. The data is typically used to maintain context and state information.

A rule conditionally maps one or more pieces of data to a set of actions to perform if the conditions are met (or not met). The IF portion of a rule defines the condition. The “THEN” portion defines the set of actions to take when the condition evaluates to TRUE, and the optional “ELSE” portion defines the set of actions to otherwise take. The rule defines how data will change based on the current state of the data. For example:

```
Rule temp_warning is:
if  current_temperature > Max_Temp then
{
      color = “red”.
}
else
{
      color = “green”.
}
```

The name of the rule is **temp_warning**. In the condition clause, the data item **current_temperature** is compared to another data item called **Max_temp.** When the comparison evaluates to TRUE then the **color** data item is set to the value, “red”. Since **color** is a data item, other rules could depend on it. Those rules may subsequently be processed.

All rule processing is handled by a software package called a “rule engine”. As data changes and external events occur, the rule engine determines which rules are eligible for firing because their condition clause becomes true. These rules are placed on a list called the Agenda. Once all of the eligible rules have been determined, the rules on the agenda are processed, or “fired”, one-by-one. Note: once a rule fires, other rules may be added or removed on the agenda as a result of changes incurred by the just fired rule. The processing continues until all of the rules on the agenda have been processed. The order of processing is generally not defined. This general rule and agenda processing is known as the RETE-Algorithm, and it is probably the most common algorithm used by rule engines today. The algorithm is fast and efficient.

Expert Systems generally excel at handling situations where the current state of the data is incomplete or unknown, which is perfect for a multi-parameter proposal development effort. A variable can be compared to a special case value such as “Unknown”. Rules can be defined that depend on missing data. They typically prompt the user to ask for more data.

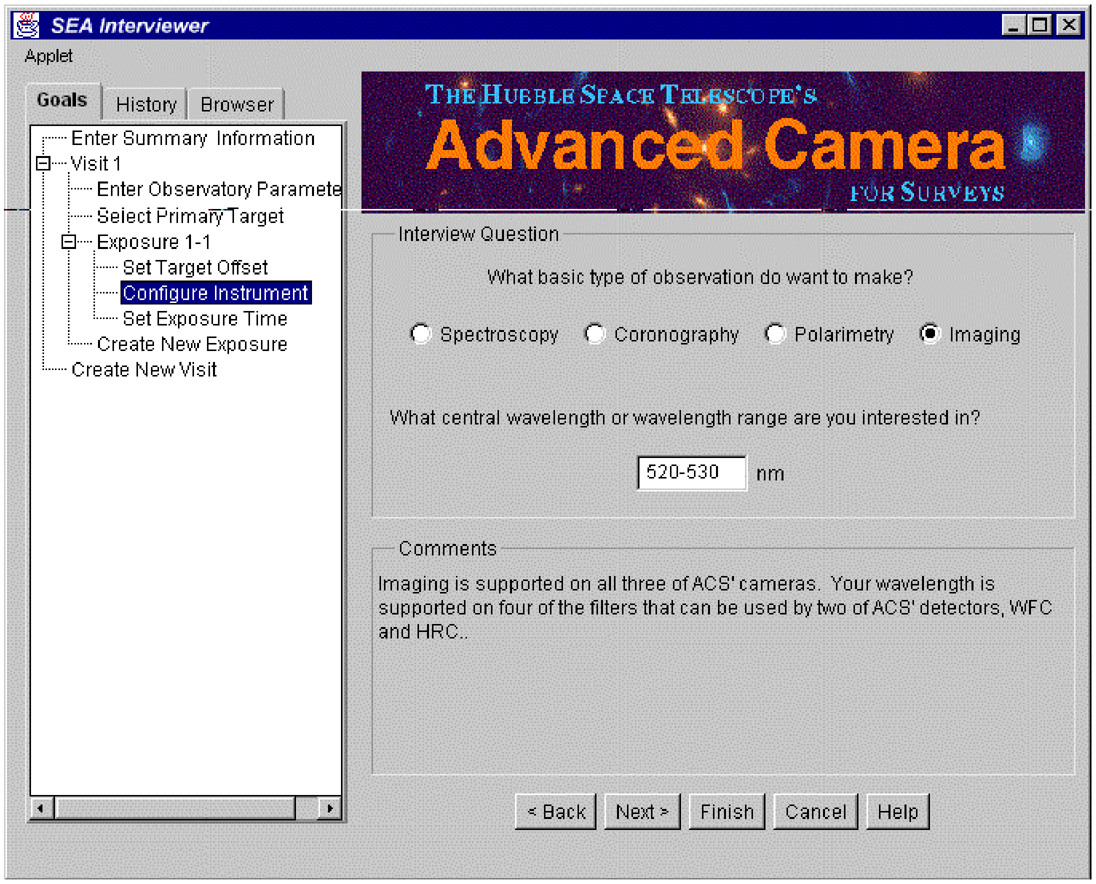
The expressed purpose of Expert System Technology seemed to match well with the purpose for the SEA. We hoped to capture at least some astronomical domain knowledge of various expert users and build into the SEA. The SEA could use that knowledge to help the user in a variety of ways.

5.2.2 First Attempt-Interview Mode

The original idea was to use an Expert System as the underlying foundation of the SEA. It would work behind the scenes to monitor the proposal development process. The general paradigm was modeled after the tax form interview mode in a familiar program such as MacInTax or TurboTax . In those programs, the user interacts with an application that is an “expert” in the income tax preparation domain. As we are sure you are familiar, income tax forms are complex with a great amount of data and many intertwining rules and special cases.

The income tax programs have two primary modes of operation. First, they provide direct access to the forms, letting the user directly control what to do next. The programs handle the propagation of values and calculations as needed, much like a spreadsheet. The second is an interactive Interview mode. The programs guide the user through the potentially lengthy tax preparation process through question and answer sessions. The determination of the next step is based on the user's answers to previous steps combined with the domain knowledge of tax preparation process. The programs ensure the data provided by the user is consistent and "makes sense".

We initially viewed the proposal development process to be similar to the tax form preparation process. The SEA expert system rulebase would present one or more questions to the user, monitor the user's responses, then generate appropriate (insightful?) questions and provide feedback in the form of comments. The expert system would act as a guide through the process. Figure 5 below shows an example of an interview page.



What basic type of observation do want to make?

Spectroscopy [✓ Imaging] Coronography Polarimetry

What central wavelength or wavelength range are you interested in?

520-530 nm

Imaging is supported on all three of ACS' cameras. Your wavelength is supported on four of the filters that can be used by two of ACS' detectors, WFC and HRC..

Figure 5: Interview Mode

Our prototype of this paradigm was not very successful due to a variety of reasons:

- The somewhat rigid interview sequencing tended to hinder exploration. The user had to follow prescribed paths through the interview process.
- The questions often didn't fit well with what the user was thinking. The users were generally focused on the science, not on forms entry or the structure of a proposal. It was often difficult to anticipate what the user might want to do next.
- An interview paradigm was not appropriate for every portion of the proposal development process. For example, the user might want to set an exposure time using the Exposure Time Calculator (ETC), not answering the next interview question.

- The underlying expert system engine technology was immature, resulting in a single monolithic and overly complex rulebase. The rules became intertwined with lots of flags and conditionals to control the flow of rule execution and to maintain processing state. As SEA grew, the rulebase became unwieldy and slow. The rulebase would only get more unwieldy, slower, and more complex as new features and capabilities were added to the SEA.

The all-inclusive Interview Mode was abandoned during the early stages of the SEA prototype development.

5.2.3 Second Attempt – The Assistant

We dramatically scaled back the scope of the expert system technology in our second integration effort. Rather than use the expert system as a primary foundation technology underlying all of the SEA, we relegated the expert system to a small portion of the application. Specifically, we used the expert system as an “assistant” to help the user with exploration, for example, which is the best instrument/detector/filter combination for my scientific needs? The expert system only became active when the user selected the "Assist" button on the Instrument Configuration panel. In other words, the user had to request the assistance from a dormant expert system. Figure 6 shows the first assistant panel.

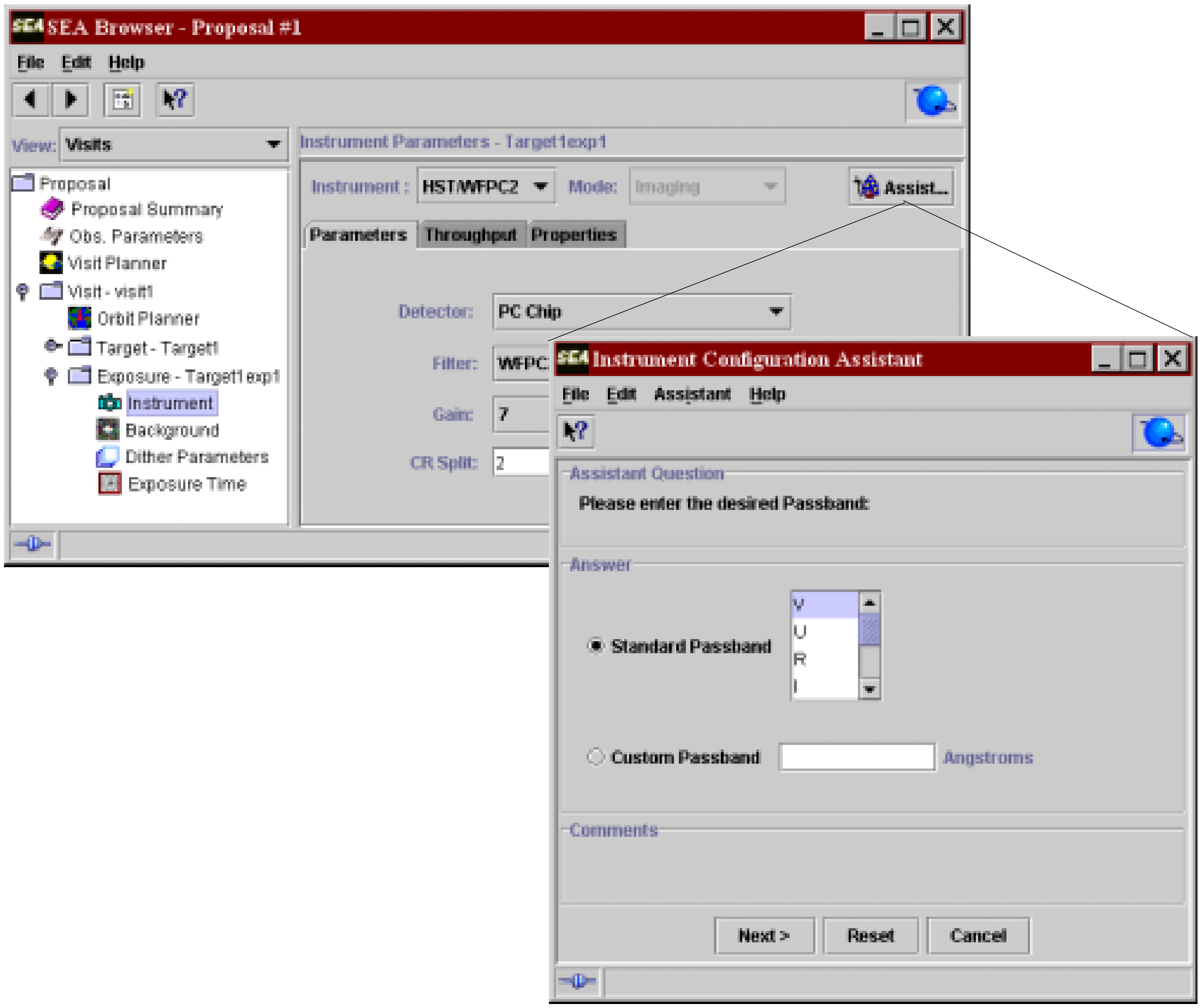


Figure 6: Instrument Configuration Assistant

The expert system was idle most of the time and only became active when requested by the user. The Expert System would wake up and prompt with two questions: (what passband? as seen above and what type of observation?). The Expert System would traverse through its list of combinations in its knowledge base and assign a "goodness" factor to each of them. The “goodness” factor was a numerical value computed from the filter’s effective wavelengths and how well they compared to the specified passband. The combinations were sorted by their goodness factors and presented to the user for selection. This strategy was highly appreciated by many of the evaluators as they saw its use during “Phase 0”, the exploratory stage of the proposal preparation process. For many other users who focused their evaluation of SEA as a Phase II tool, this strategy was considered okay but not useful.

The rulebase became very small and algorithmic due to its limited responsibilities and the mechanical way we chose to process the filters. During the rulebase development, the vendor supplied rule engine became more powerful and better able to control rule-processing flow. We no longer needed a lot of data to maintain the state and to control the rule flow execution as we saw in our first integration attempt (see section 5.2.2). The reduced complexity caused a complementary reduction in rulebase size.

The SEA is implemented in the Java programming language. One of the built in features of Java is support for application threads, one or more parallel execution paths within a single application. Figure 7 demonstrates how the Instrument Configuration Assistant thread was implemented within the SEA.

The sequencing is as follows:

- The user requests assistance by selecting the **Assist** button on the Instrument Configuration panel.
- The rulebase resolves Unknown data by asking the user for additional input.
- The rule engine applies the rulebase rules.
- Results are loaded into a Results array.
- The user views the Results array and usually selects one of the choices.
- The Main SEA thread continues.

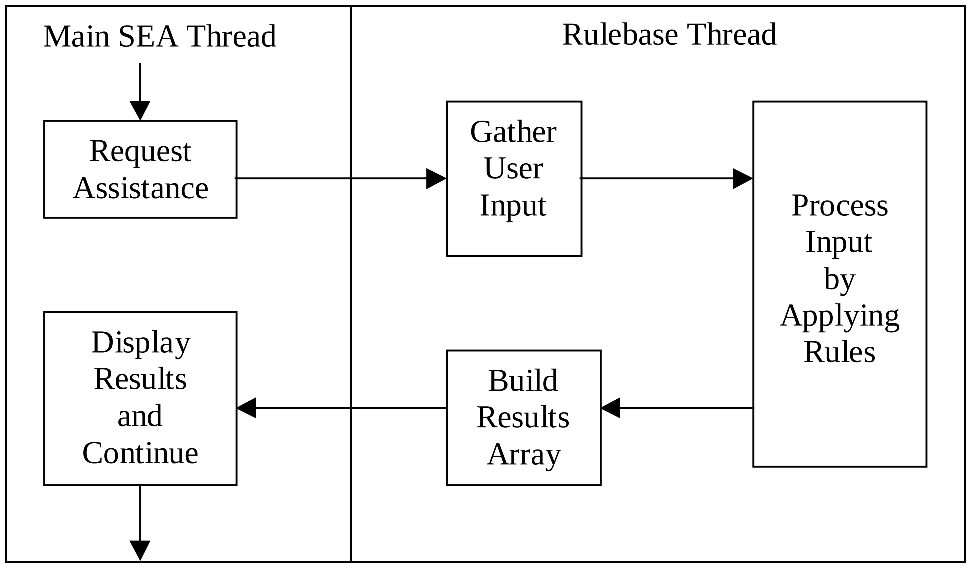


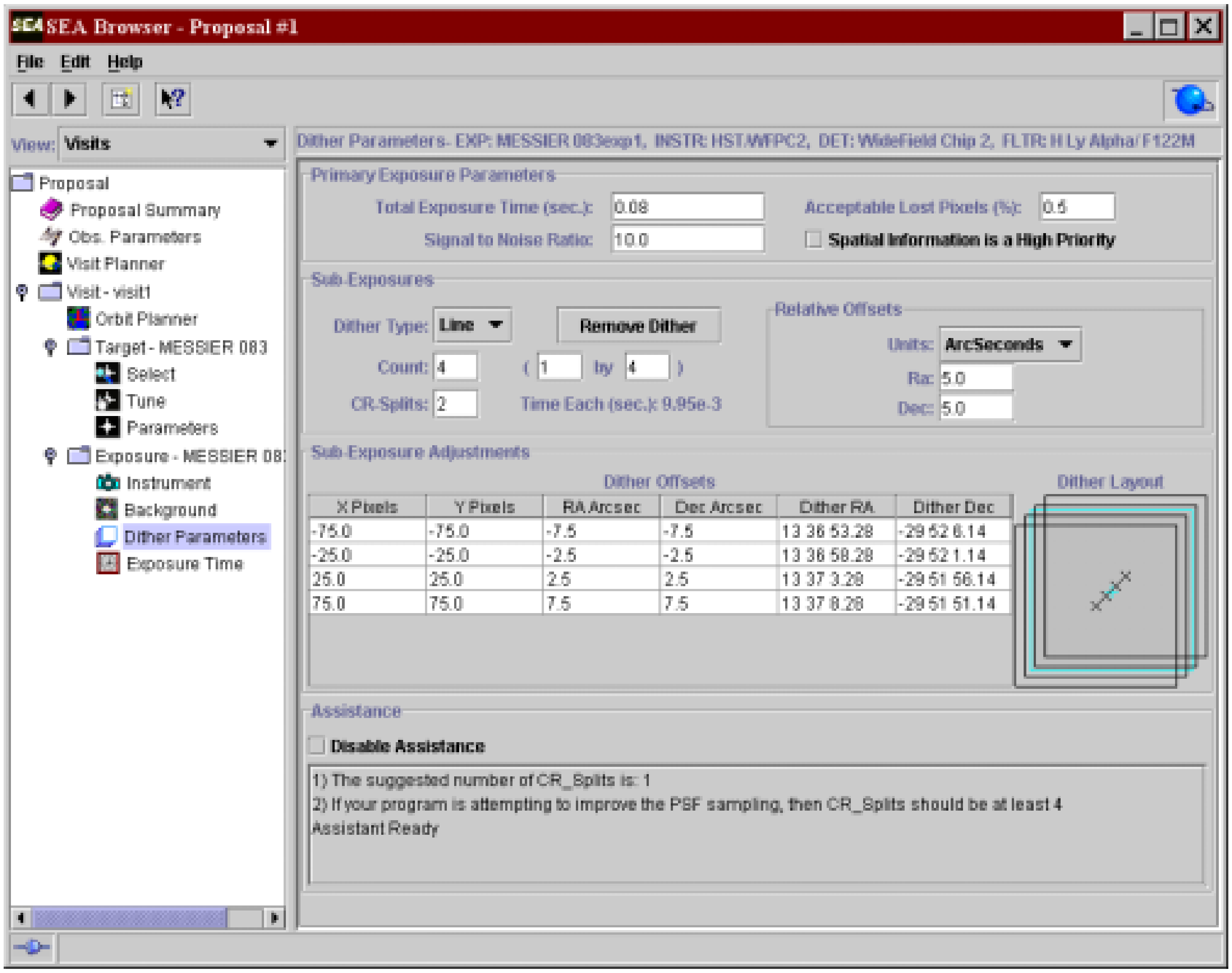
Figure 7: Instrument Configuration Assistant Rule Processing Flow

This processing is not a very effective use of threads. The above processing could just as well have been written in straight Java code and handled as a sequence of method calls, thereby reducing the runtime overhead for rule processing. Rulebases are loaded and interpreted at runtime by the Rule Engine. The Rule Engine we used is a large and complex collection of Java classes and it often takes many seconds to load the classes and run the rule engine. Subsequent executions are much quicker because they do not incur the class loading and rule compiling costs, but that initial load can be frustrating for the user.

While the Assistant is still a part of the SEA, we felt we could do more, which brings us to our third attempt at integrating expert systems into the SEA.

5.2.4 Third Attempt – Helpful Observer

The third attempt is somewhere between the two previous attempts. One of the last features we added to the SEA was a panel for specifying an Exposure’s Dither Pattern (see Figure 8). In this module, the rulebase passively watches the user's changes and offers suggestions and warnings as appropriate. The user is free to explore while the Expert System analyzes state changes and makes suggestions in a small text area at the bottom of the module. This attempt is similar to our first attempt, but not as pervasive and it doesn't force itself on the user. We understand that our users are typically human with a variety of human temperaments; therefore we included a mechanism for the user to completely disable all comments and suggestions. We felt it was important to avoid annoying users with too much unrequested help. Although this was the least developed of the SEA's functionalities, the evaluators thought that such a functionality would be useful once it is fully developed.



Dither Parameters- EXP: MESSIER 083exp1, INSTR: HST/WFPC2, DET: WideField Chip 2, FLTR: H Ly Alpha/ F122M

Primary Exposure Parameters

Total Exposure Time (sec.): 0.08

Signal to Noise Ratio: 10.0

Acceptable Lost Pixels (%): 0.5

Spatial Information is a High Priority

Sub-Exposures

Dither Type: Line

Count: 4 (1 by 4)

CR-Splits: 2

Time Each (sec.): 9.95e-3

Relative Offsets

Units: ArcSeconds

Ra: 5.0

Dec: 5.0

Sub-Exposure Adjustments

Dither Offsets

X Pixels	Y Pixels	RA Arcsec	Dec Arcsec	Dither RA	Dither Dec
-75.0	-75.0	-7.5	-7.5	13 36 53.28	-29 52 6.14
-25.0	-25.0	-2.5	-2.5	13 36 58.28	-29 52 1.14
25.0	25.0	2.5	2.5	13 37 3.28	-29 51 56.14
75.0	75.0	7.5	7.5	13 37 8.28	-29 51 51.14

Assistance

Disable Assistance

```
1) The suggested number of CR_Splits is: 1
2) If your program is attempting to improve the PSF sampling, then CR_Splits should be at least 4
Assistant Ready
```

Figure 8: Dither Pattern Specification Panel

Like the SEA, the underlying Expert System Technology also advanced since our previous attempts. Two key improvements enabled us to create a more effective implementation.

- Vastly improved asynchronous event handling. The Java rule engine we used supported rules firing when an instance of an object was created, deleted, initialized, or when specific properties of an object changed or was required. In addition, the main Java SEA application could be notified when the rule engine created new objects (see Figure 9). The combination of all these asynchronous event mechanisms enabled the rule engine to monitor events occurring in the main SEA thread and to report rule processing results as they happened.

- The rulebase became much more modularized and efficient. In previous versions, all rulebases had to be a single monolithic rule file. All rules were stored together and evaluated together. This was very inefficient and hard to program. The improvement was the implementation of something called a “ruleset”. A ruleset is a grouping of rules into logical collections. The rulesets enable the programmer to control which groups of rules to apply for a given context. Only those rules necessary in the current context need to be evaluated. By segregating rules that do not apply, you avoid possible unintended interaction among all the rules and you increase efficiency.

These capabilities helped to eliminate the monolithic rulebase problem as seen during our first attempt. In addition, since we didn’t force a dialog with the user we avoided the exploration problems also seen in our first attempt.

When the Dither module first starts up, it creates a separate rulebase thread. This new thread loads the rule engine Java class code and the Dither rulebase. While that thread is initializing, the main Dither module continues in the main thread to build and display itself. The Dither module then accepts user input. The rule processing flow is as follows: (see Figure 9 below)

- Every time the user changes a value in the Dither module, a new DitherContext object is created and passed to the rulebase thread.
- The Dither module continues to accept other input.
- In the meantime, the rulebase thread extracts the necessary information from the DitherContext and applies the Dither rules to that data.
- During processing, one or more DitherResult objects may be created by the rulebase.
- Every time a DitherResult object is created, it is asynchronously transferred back to the main thread. The main thread extracts and displays the result information.
- Once all rule processing has completed, the rulebase goes back to step 1.

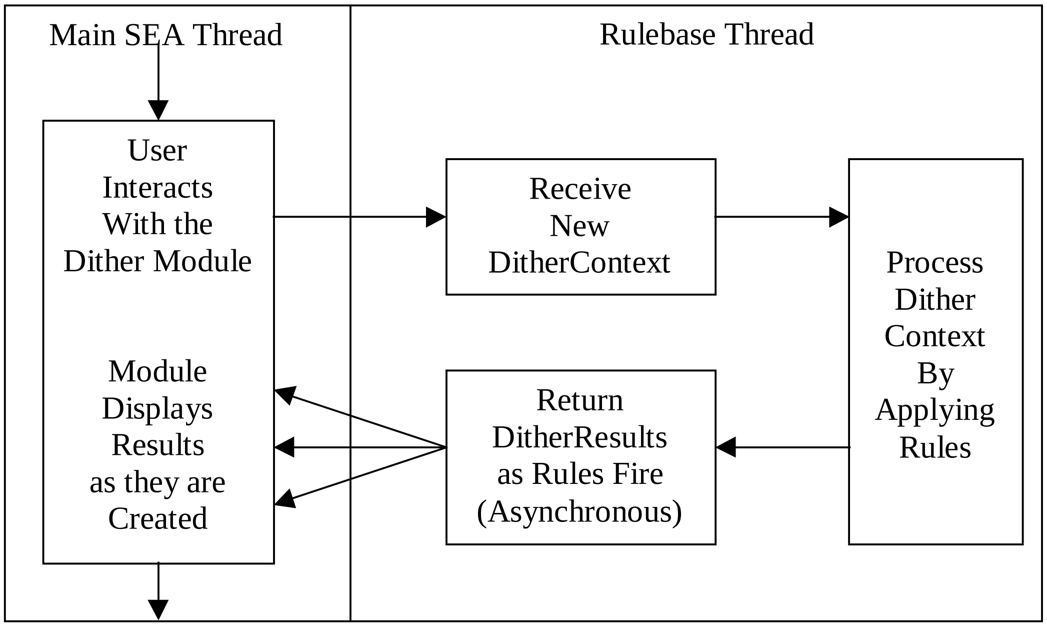

Main SEA Thread
Rulebase Thread
User Interacts With the Dither Module
Module Displays Results as they are Created
Receive New DitherContext
Process Dither Context By Applying Rules
Return DitherResults as Rules Fire (Asynchronous)

Figure 9: Dither Pattern Rule Processing Flow

The difference between this processing flow and the one in Figure 7 is that the main thread continues to operate while the rule engine is working. At any time, the rule engine can communicate any desired information back to the main thread simply be creating a new DitherResults object. The Dither rulebase is essentially a controlled passive observer of the main Dither module. It can provide guidance only when permitted to do so, and only in a quiet and unobtrusive way.

5.2.5 Lessons Learned

Contrary to what expert system vendors say, rulebases need to be written by programmers. Rulebases use program constructs (variables, conditionals, subprograms, etc.) and are often interrelated and require a great deal of design and planning. Individual rules may be easy to define but they are often hard to integrate into a coherent rulebase. We had originally thought that the scientists could update these rules themselves, but this soon became unrealistic.

We often had a difficult time determining whether a rule should be implemented as a rule in a rulebase or as normal Java code. Even after writing a few rulebases, the answer to this is still not obvious.

Overall, expert system technology shows great promise. It is expected that expert systems will have an expanded role within the SEA. We feel the combination of the Helpful Observer along with the Assist button technique demonstrates good uses of expert systems. The support technology now exists so that in the future we can continue to expand the role of the expert system into such areas as: improved context sensitive help, intelligent automatic optimized exposure layout processing in the Orbit Planner, and efficient and effective orientation and mosaic pattern techniques in the Visual Target Tuner.

5.3 User Interface Challenges

The SEA design required a display that represented the contents of the proposal to the user. This user interface (UI) would display the high-level proposal elements along with operations to perform on those elements. This UI would form the foundation for navigation between the various tools. From a computer scientist's view, the obvious answer to this problem was a hierarchical view. After all, an HST proposal is composed of a number of Visits, each of which has a Target and one or more Exposures. So the original tree navigation view was born. We quickly realized, however, that this was not the most user-friendly presentation. Indeed, *users often have difficulty with the tree view*. One reason for this is that the tools are hidden within the tree, which means that the user must drill down into the tree to discover the tools.

Despite the fact that the tree is an imperfect design, we have yet to find a superior design. One refinement of the tree design, however, has at least been a minor improvement. We discovered that users approach the problem of observation definition from different directions. Some users work from a top-down approach, defining their visits, then exposures, while others wish to define their exposures then assign them to visits. Other users prefer to define their targets first. To accommodate these different views, we added the ability to reorder the tree into different "views" on the proposal. The user has the ability to quickly reorder the tree to highlight Visits, Exposures, Targets, or Instruments.

Another interface challenge has been how to handle the different platform assumptions of individual users. Since the SEA is a platform-independent application, a decision had to be made regarding which platform features to emulate. Our goal was to make the SEA look and feel like a modern Microsoft Windows application. Windows was chosen instead of Unix/Motif simply because Windows contains many more useful user interface concepts (even if they did not originate in Windows). The idea was that if the SEA was similar to other tools with which the user was already familiar, the initial learning curve would be less. However, we were concerned that users who were not familiar with Windows (still a sizeable portion of the scientific community) would have difficulty with some of the standard Windows concepts. These concerns were realized in the evaluation. The ability of a user to intuitively grasp features like Tool-Tips, drag and drop, and multi-selection, seemed to correlate with the user's Windows (or Macintosh) experience. *When* *designing a platform-independent application, one must consider that many users may not be familiar with the "standard" features of another platform's user interface.*

The original user interface design for the SEA was very different from the current design. An original design goal was to accommodate two categories of users, novice users and experienced users, through two different high-level user interfaces. The browser UI would provide flexible access to every element of the proposal and was intended for the experienced user. The novice UI was designed to provide a simplified means by which the software would walk the user through a series of questions that filled in the details of their proposal. The challenges that we faced in developing this "Interview" user interface are described in the section on Expert Systems (see section 5.2).

5.4 XML

The SEA uses XML for proposal storage and for configuration and preference files. Originally, we stored proposal information by implementing the Java Serializable interface. Unfortunately, implementation on our rapidly changing prototype fell behind resulting in data files that could not be reloaded.

The solution to our proposal serialization problem came from the Koala Object Markup Language project (http://www.inria.fr/koala/XML/serialization/) and their XML serialization tools. Koala's serialization API uses reflection in addition to the Serializable interface to serialize the graph of objects that form an SEA proposal. It then reforms the resulting data stream into XML. Furthermore, Koala provides an API that uses the SAX parser to reconstruct the graph. All this is performed without having to write explicit serialization methods. While we are able to use the API to read and write proposals, the Koala serialization does not result in very human readable code when applied to our data model. *A more human readable XML format that would allow users to read and update proposals outside of the SEA should be considered in the future.*

Storing user-settable preferences was a different matter. We initially used our own text-based format for preference storage that resembled C code. While readable by software developers, we decided a more user-friendly format would be desirable. Editing the original “JavaCC” files by hand was also tedious and error prone. Switching to XML allowed us to take advantage of the multitude of XML editors and parsers, and reduced our level of maintenance effort while increasing the readability of our preference files.

5.5 Interfaces to Legacy Systems

Since many institutions already have collections of programs for providing technical information, for example target visibility, optimal roll angle, guide star availability, etc., for the HST observatory, we decided interfacing to legacy systems would be a worthwhile experiment. Instead of rewriting code for target visibility, we decided to interface to the Space Telescope Science Institute's Spike scheduler.

Unfortunately, Spike is a Lisp program running on Solaris. Hence, instead of porting Spike to MS Windows we decided to use Spike's socket interface that gave us the ability to directly connect to a Spike process and query it for data. Further, instead of writing socket calls to Spike directly from the SEA application, we wrapped Spike in a package outside of the SEA called JSpike. This provides the SEA with a generic interface for accessing Target Visibility Data whether online or from data files without needing explicit knowledge of the scheduler.

Further abstraction will allow us to access other legacy systems that implement our scheduler interface. By separating the legacy components into individual JAR files we can provide easy substitution of legacy system components by simply replacing JAR files.

The disadvantage of depending on legacy systems is that they can be the “Achilles heel” of the system. The SEA implements its HST-specific exposure time calculations by connecting to the SYNPHOT package at the Institute. While this was the correct decision from the standpoint of reuse and development time, the SYNPHOT connection was the least reliable aspect of the SEA. If SYNPHOT was down, or was in an unstable state, or the network was down, the SEA's exposure time calculation was crippled. The lesson we drew from this was that while interfacing to legacy systems is generally a good thing that can save quite a bit of code rewriting, one must consider how the system should operate when the legacy resource is not available. Complete dependency on an external legacy resource should perhaps be avoided if possible.

6 Evaluation

During February 2000, we conducted user evaluations on SEA. Our evaluators were chosen from astronomers with accepted HST Cycle 9 programs that focused on HST's WFPC2. This evaluation was to determine if we had succeeded in achieving our objectives of having an intelligent, intuitive, distributed, adaptable, integrated, and flexible system. We were also interested to learn how usable the SEA system was.

6.1 Evaluation Methodology

6.1.1 Evaluator Profiles

Our original plan was to have at least 21 evaluators so that both the evaluators and programs would cover a range of user expertise and range of program types (see Table 1), and allow us to answer the many questions in a "semi-statistical" manner.

Table 1: Choice of programs to be considered for evaluation

PI expertise level	Type of proposal	Size of proposal
New PI	Simple point and shoot programs	small with 1-2 targets
Moderately experienced	Programs using some special scheduling requirements	large > 10 targets
Expert	One program which would be considered tough for various reasons during the strategy phase	

Due to time and budget constraints we chose 13 principal investigators (PI; 8 from within the Baltimore Washington area and the remaining 5 from the Los Angeles area) from the Cycle 9 accepted pool for WPFC2. We obtained background on them by distributing a questionnaire to each of the prospective evaluators. Our final choice of evaluators included post-doctoral fellows to senior faculty and spanned the full range of familiarity with HST's Remote Proposal Submission process (RPS2), i.e., very familiar with RPS2 (have used it for more than three proposals) to not familiar (have only seen colleagues use it) (see Figure 10).

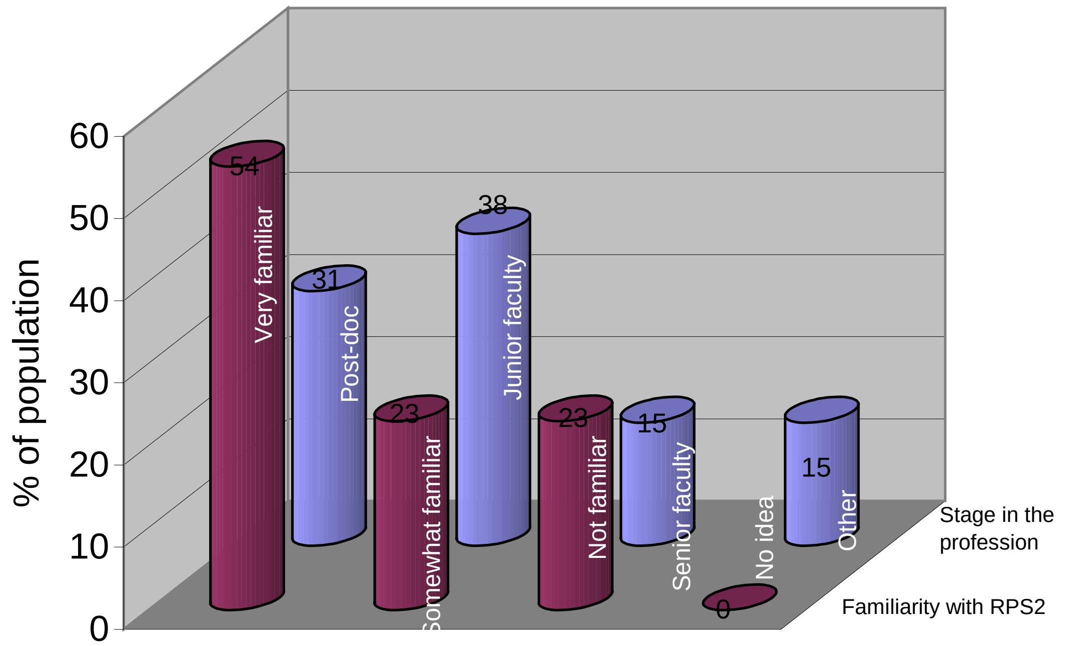


Figure 10: Evaluation Profiles

6.1.2 Evaluation Elements

We originally planned two evaluation phases: first, our evaluators would run some test scenarios that would familiarize them with SEA's features; second, the evaluators would use SEA to develop their phase II Cycle 9 proposal as much as they could given SEA's current feature set. We quickly learned that our user community was resistant to committing a large block of time upfront for the evaluation. We adjusted our plan to reduce the planned time to 2 hours (although some evaluators actually spent up to 5 hours with us on the day of the evaluation). To maximize the feedback with only a 2-hour evaluation window, we tried to accommodate each evaluator's scientific interests as we introduced them to the SEA. We focused on ensuring that all evaluators were introduced to key functionalities that they were to evaluate, but had to eliminate the second phase of the evaluation (attempted generation of a full Phase II proposal).

We had a well-defined set of tasks for the evaluation, but soon found out that most evaluators did not want to follow that strict path. They wanted to freely move within the SEA tool and try out their favorite observing strategies. Since SEA was designed to accept different proposal preparation strategies, we adapted the evaluation plan for each user. We tried to accommodate each evaluator's scientific interests as we introduced them to the SEA.

During each evaluation, the evaluator did all the computer input, and was observed and coached as needed by a team that consisted of the SEA evaluation coordinator, an HST support staff observer, and an SEA developer. The coordinator guided the evaluator through a pre-defined set of tasks designed to span SEA's capabilities. The coordinator provided coaching as needed but tried to allow the evaluator time to discover the solution on their own. The support staff observer and developer took notes on the results, process and reactions of both the evaluator and SEA. After completing the online tasks, our evaluators completed a survey that asked them to rank several features of SEA in both quality and value.

The evaluation plan also included some amount of usability testing with the objective of providing a means for the targeted user community to provide feedback on the SEA to the software developers.

The evaluations were conducted at the evaluators home institute so that we could also understand platform and connectivity issues. Our evaluators were mostly using the Solaris platform, but we also did some evaluation on the Windows and DEC Alpha platforms. We did not run into any connectivity issues. The speed of installation depended on the memory in the machines. The InstallAnywhere software by ZeroG Software worked well. Prior to our evaluations, we had never tried to run SEA on a DEC Alpha, and Java's true platform independence was tested in-situ. In this case we crashed when the expert system was invoked, but otherwise SEA performed nominally on the DEC Alpha.

The evaluation could not be done electronically since we wanted to understand any relevant behavior or comments made by the evaluators as they prepared their test proposals. After completing the online tasks the evaluators were asked to complete a survey, the results of which are in the next section, where they ranked several features of SEA in both quality and value. With the above described evaluation strategy we were able to draw some conclusions about how users construct their observing programs, and the level of expert information that they need.

An important output of our evaluation is that we saw a number of experts use different strategies in how they translate a scientific idea into a proposal. The SEA developers all took part in the evaluation process and unanimously found the experience valuable. In a traditional software development scenario, developers rarely get to spend quality time with users. We concluded that this kind of interaction between developer and user is better than the more traditional restricted method where developers and users interact primarily via a set of formally approved requirements. Our evaluation gave greater insight into how the end-user intuitively tried to use the tools.

6.2 Evaluation Results

6.2.1 Evaluation of SEA features

The following results were obtained when users were asked to rate the SEA features/functionality on a scale of 1- 5, where 1=excellent, 2=above average, 3=average, 4=below average, 5=poor. .

Table 2: Evaluation of the Visual Target Tuner Module:

Feature or Functionality	**Grade**
1. Ability to display the Digitized Sky Survey	1.2
2. Ability to display the HST field-of-view on the Digitized Sky Survey image, to see the footprints of all the instrument apertures simultaneously	1.2
3. Ability to graphically and dynamically manipulate any of the HST instrument apertures	1.3
4. Ability to display the guide star catalog and then filter the catalog objects dynamically	1.5
5. Access to the various archives such as NED or SIMBAD to obtain target coordinates and other relevant information regarding the field-of view something the matter with cell size and font size here.	1.5
6. Ability to capture scientific constraints and then optimize a feature, e.g., the include/exclude feature that can be used to determine the range of available orientation angles	1.9
7. Availability of image options such as the color tables etc.	2.0
8. Availability of analysis features, e.g. centroiding on a star	2.0
9. Availability of canned features, e.g. mosaicing a region	2.5
10. Ability to access the Digitized Sky Survey via a batch process, so that multiple images can be retrieved in one call	2.5

11. Ability to have 3-dimensional image manipulation	3.3

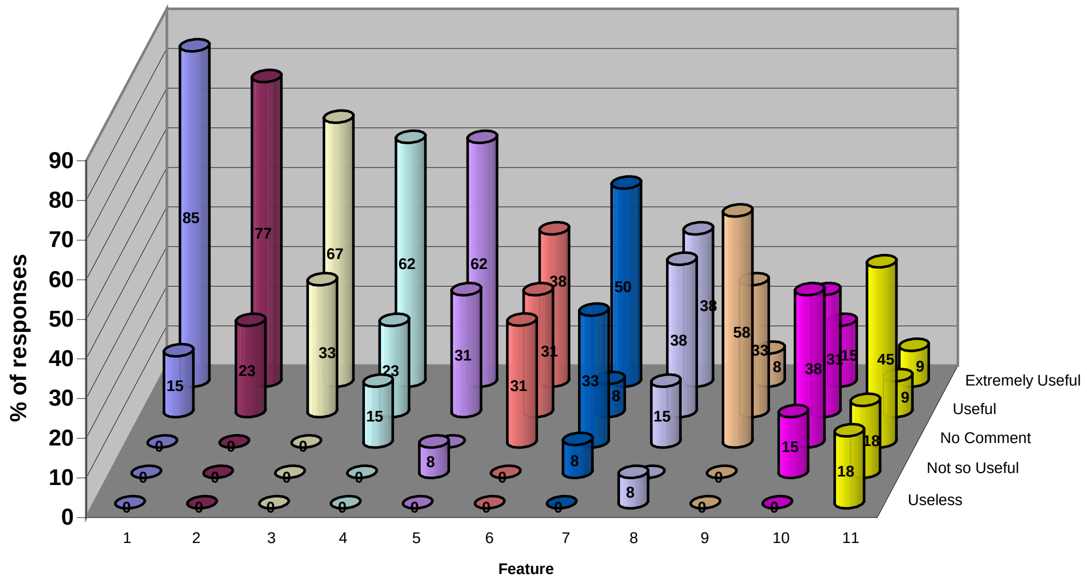


Figure 11: Visual Target Tuner Features versus Grade

Table 3: Evaluation of the Exposure Time Calculator Module:

Feature or Functionality	Grade
1. Ability to access and manipulate multiple parameters (target/instrument) simultaneously	1.4
2. Graphical access to many different spectral energy distributions	1.5
3. Graphical access to operational instrument response curves, and ability to compare these with the object's spectral energy distribution	1.5
4. Ability to compare two sets of exposure parameters	1.9
5. Ability to compare multiple instruments simultaneously	2.0
6. Ability to manipulate the graphics display parameters	2.3

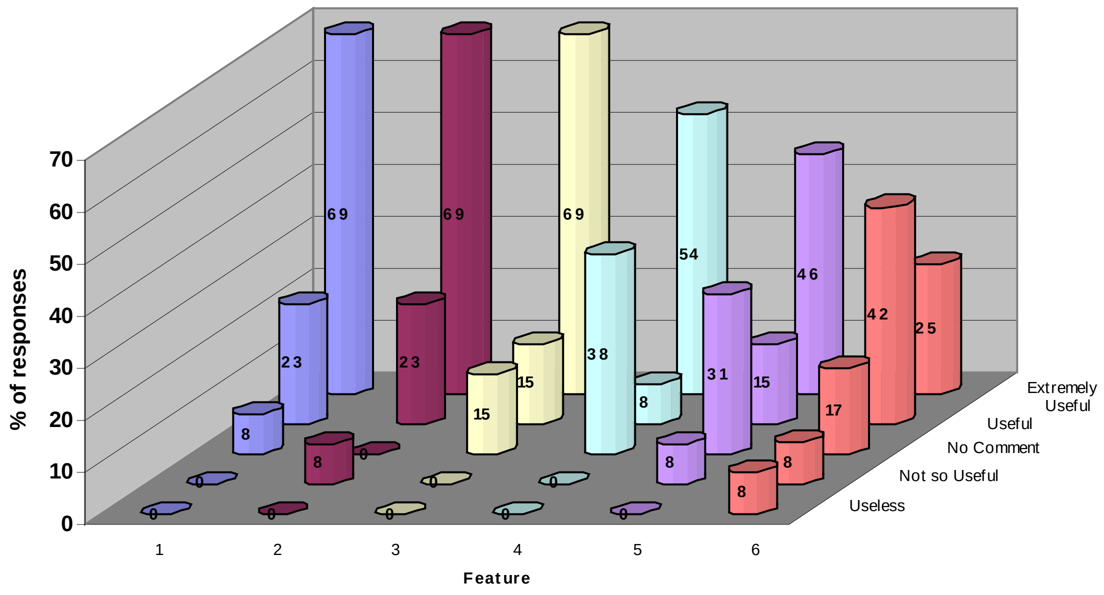


Figure 12: Exposure Time Calculator Features versus Grade

Table 4: Evaluation of the Orbit Planner and Visit Planner Modules:

Feature or Functionality	**Grade**
1. Graphical display showing detailed overhead information etc. of all the exposures in a visit. Graphical display of all visits in a proposal.	1.3
2. Ability to graphically and dynamically manipulate exposures in a visit to optimize the scientific returns from that visit.	1.6
3. Ability to place constraints on the exposures and visits in a more “natural language” way and then to see which exposure/visits are connected to each other with constraints	1.6

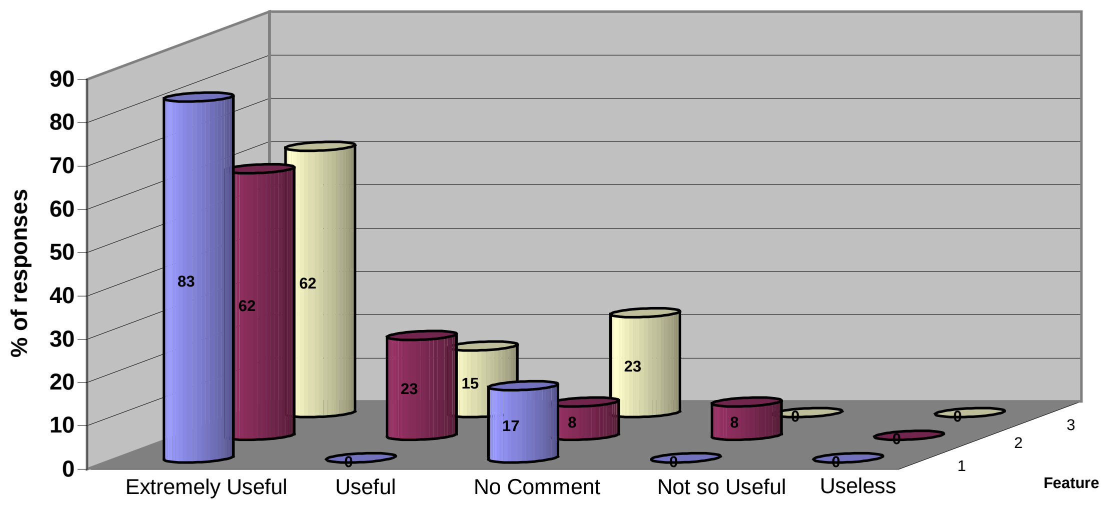


Figure 13: Orbit Planner and Visit Planner Features versus Grade

Table 5: Evaluation of the General Features in the SEA:

Feature or Functionality	**Grade**
1. Instantaneous updates to proposal parameters where possible	1.6
2. Graphical interface where possible	1.8
3. Random access to any part of the proposal, i.e., no particular order to developing a proposal	1.8
4. Ability to develop a proposal interactively, or directly via tables using the multiple table views	1.8
5. Access to handbooks/reference material	1.9

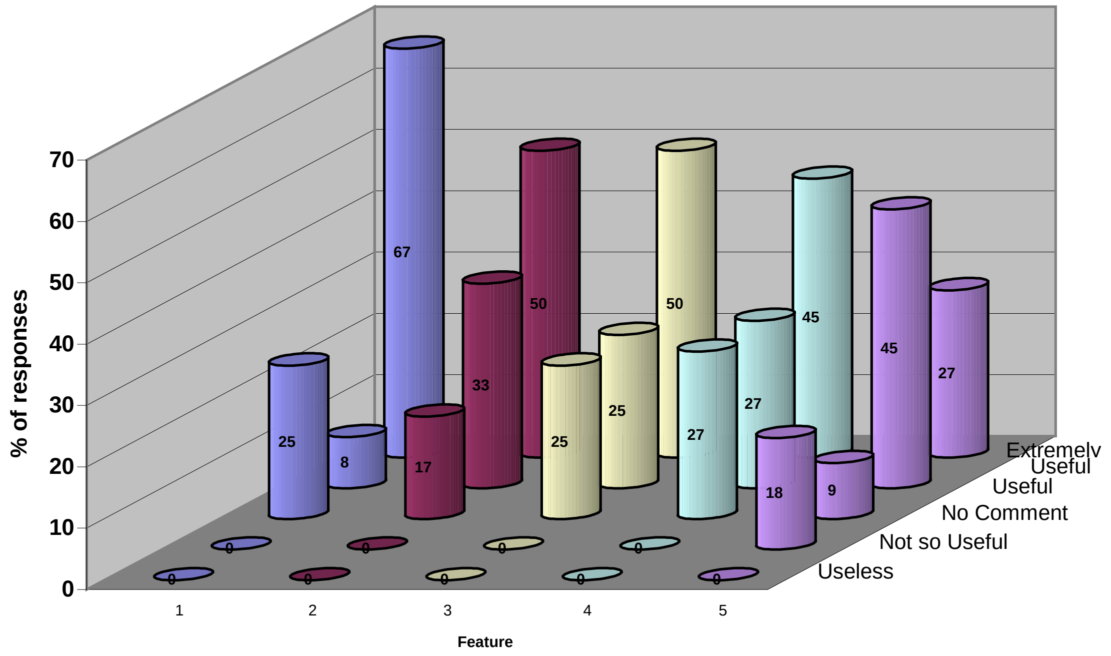


Figure 14: General Features versus Grade

6.2.2 How did users perceive the SEA software development philosophy

- The system should be *intelligent*: Our attempts at providing expert help were appreciated, but it was felt that more work needed to be done before these ideas were ready for operational use. We feel that this users opinion does not indicate that an intelligent system is not appreciated, but rather indicates that the expert system modules are not yet mature for operational use.

The system should be *intuitive*: Users found that the user interface needs to be improved to be more intuitive. They all indicated that an “interactive manual” or manual would be useful to help users understand how to work with SEA. The user interface issue is very important to solve (see section 5.3). With regards to the need for a manual we feel that the flexibility/non-linearity provided in SEA makes the features and modules more complicated than they are.

- The system should be *distributed*: All the users appreciated the ease with which SEA could be downloaded.

- The system should be *adaptable*: This feature could not be evaluated, but the modularity of SEA allows the software to be adaptable.

- The system should be *easily integrated*: This feature could not be evaluated.

- The system should be *flexible*: We have tested the flexibility of the system by determining how easily it can be adapted for other observatories (see section 8.4).

Users ranking of software guidelines that are useful from the user’s perspective are shown in the following table. In this table the grades are as follows: 1=very important, 2=important, 3=no opinion, 4=not important, 5=useless

Feature	Grade
Easy use - To accommodate user expertise and preferences, proposal information can be accepted in multiple formats.	1.5
User orientation - All components of the system will use terms and concepts which are meaningful to astronomers.	1.9
Responsiveness and speed - Whenever possible, results of user actions will be available immediately. Instantaneous graphical updates when changes are made will be presented to users whenever possible.	1.9
Uniformity - All tools will use consistent terminologies and have a similar "look and feel" to reduce the learning curve.	2.0
Scientific feedback - The system will provide information needed to make scientific trade-offs. The impact of a choice will be shown in a meaningful way, and thus users will be made self-sufficient.	2.1
Easy installation - Straightforward web-based installation with a highly portable, platform-independent implementation.	2.1
Interoperability - Tools will be able to share information, alleviating users from having to manually enter and re-enter data and re-process information.	2.1
Common environment - Observers will have access to the same tools environment and configuration as observatory staff.	2.1
Useful documentation - Documentation will be an integral part of the toolset and will be structured to allow efficient access by humans *and* software tools.	2.5

6.2.3 Highlights of the Evaluation

In general, do users perceive the SEA as better than the current HST Phase II proposal creation process and associated tools?

We can with confidence say that visual, interactive tools were found to be highly useful by the users for exploring the multi parameter space. It allowed them the ability to visualize their scientific needs better and they did not have to concentrate on observatory dependent technical aspects of observing.

All evaluators (except one) felt that the SEA user experience was far better than that provided by the currently available HST Phase II tools such as RPS2. They also cautioned us that SEA did not have all the features necessary to develop a complete HST proposal and that some improvements are necessary to make proposal development in SEA more intuitive.

Did the on-line help, context-sensitive help assist the user?

The concepts presented were good, but present help is not extensive. This feature will be extremely useful when the help pages in the tools are fully populated with how to use the tool and with observatory specific technical information. In the prototype we just attempted to show how such a feature could be used by integrating the HST call for proposals with the software tool and by having a limited help on the features in the tools.

Do the tools provide relevant scientific assistance to the user in the form of wizards?

The evaluators found that wizards would be useful and commented that the present wizards need to be further developed and new ones added. They all felt that they would be comfortable if the system provided solutions for a given observing strategy, but they cautioned that they needed to have full control to override the solutions at anytime. Further, the algorithm used to find the solutions should be clearly indicated. We also asked the users to identify features/modules that they would like to see further developed. The answers to this question were very HST specific, but canned observing strategies where one could automate the process of applying customizable observing strategies to observing programs was highly appreciated.

Does the environment provide easy access to technical and reference material?

Yes, access to information was appreciated. We found that easy access depended on how familiar the evaluator was with the windows platform as we have heavily used the windows features as standard for the SEA tools. In terms of technical information, users appreciated access to reference material and recommended that such access to information on schedulability of a program was essential.

Did observers focus on science?

Yes, visualization allowed users to concentrate on scientific issues and not on the non-value added issues.

Do the tools accommodate different levels of users?

For flexibility, proposal development in SEA does not force any particular order. Some users valued this flexibility; others were frustrated by the apparent lack of focus and direction.

6.2.4 What else did we learn?

- Users perceive certain concepts very differently. For example, context sensitive help and access to reference information was appreciated, but when we asked the question "Would you like documentation to be integrated with software" it ranked as a very low priority. Users did not realize that without integration of documents with tools, context-sensitive help would not be easily available. By evaluating the two questions differently, they were stating that they still liked to access documentation in book format, even if they are reading the documentation electronically.
- We had anticipated that our better-developed, more mature features/modules would get a better reception, but we were surprised at the high correlation between the maturity of a module/feature, and how important the evaluators ranked the functionality. The VTT was one of the modules in SEA with a number of features and this conclusion is clearly apparent in this module. From figure 11 we see that a feature such as aperture manipulation, which was well developed, ranked highly in comparison to a feature such as batch processing or 3-dimensional graphics ability. These features were not cleanly developed/or easily available on all platforms as they received the low evaluation grade.
- The instantaneous updating of information was seen as a two edged sword – there were occasions when this was considered essential (i.e. in the VTT), while in other instances it was found to be unacceptable (i.e. in the ETC where users often want to change several parameters at a time and then see the impact).
- Speed of any software has always been an issue with users, we have tried to keep this in mind. During the evaluations we found that this issue was never raised. We feel that this is because of the availability of interactive features and instantaneous updates.
- For much of the reference information, the SEA connects directly to servers at STScI and is highly dependent on Internet connectivity. We need to develop strategies to pull all the necessary and relevant information at the start of a session so that a user does not require network connectivity. This is especially important given that the astronomical community spans across the world and is often traveling.
- By design, SEA does not force a user to develop an observing program in any particular order. Some users quickly adapted and valued this flexibility; others were frustrated by the apparent lack of focus and direction provided by the tool
- A few of our evaluators also saw how many of these tools could be easily extended for other telescopes. They even made suggestions that we should provide users with the ability to ingest information from other telescopes into the SEA.

7 Collaboration

At the 1998 SPIE meeting on “Observatory Operations to Optimize Scientific Return" there was consensus that we explore the mutual concerns and constraints of the various observatories, so that we can understand where joint, shared, or collaborative effort might be beneficial. We therefore organized the “Workshop on Observing Tools" in October 1998[9]. Over fifty representatives, a mixture of astronomers and software developers, representing a variety of observatories (ground and space-based, optical, x-ray, infrared) attended this workshop. Our goal in organizing this workshop was to promote collaboration between observing tool development teams. It was our hope that this meeting would begin a dialogue on collaboration and software reuse that would continue after the workshop was concluded.

The meeting went quite well. It was well attended, with representatives from many different organizations including Chandra, ESO, Gemini, HST, NOAO, SAO, and IPAC. The emphasis on discussion over presentation was clearly the correct choice. Topics ranged from the commonalities between observatory needs, to the future of observing tools, to how to break down barriers to collaboration. More information on the results of the meeting can be found at http://aaaprod.gsfc.nasa.gov/workshop/.

While the meeting was declared a success, the group hoped that the meeting would be just the beginning of a larger dialogue and collaboration effort. Six working groups were setup that covered specific observing tool topics: Target Visualization Tools, Exposure Time Calculators, Defining an Observation, Optimizing Calibration Observations for Ground Based Observatories, Common Observatory Definition, and Data Services. Both short-term and long-term goals were established for each group. However, despite the best efforts of the members, none of the working groups ever produced any substantial results. Some groups started out strong but quickly fizzled out due to lack of interest or time, while others never really started at all.

While it is certainly worthwhile to strive for collaboration between organizations, it is unfortunately extremely difficult to sustain. Clearly, the workshop attendees wanted to share their efforts and work towards common solutions. But we were hesitant to push for firm pre-established “deadlines” for the groups to report back their progress, and we still failed to keep the momentum moving. Lack of organizational commitment is one reason. Another is the simple reality of daily pressures exerting priority over other "optional" activities. For a collaboration effort to succeed, it must be carefully focused so that the long-term cost of not participating is higher than the initial cost of participating in the effort. We also believe that distance discourages collaboration, if groups can make the time to get together face-to-face, much more can be accomplished in a shorter time frame.

The JSky project is one example of a successful collaborative effort. JSky is ESO's attempt at coordinating a library of reusable Java components for use in astronomy. The Visual Target Tuner (VTT) module of the SEA now uses JSky components for its underlying image rendering engine. This includes a FITS image “codec” for the Java Advanced Imaging (JAI) API. JSky enabled the VTT to utilize the JAI libraries with minimal effort, increasing the capability and performance of the VTT. The SEA team has also shared code and ideas that have been incorporated back into JSky, such as a world coordinate system library that was originally written for the VTT. JSky has been a successful collaboration for the SEA because it has actually reduced development time instead of increasing it. Still, JSky is a rather modest library, and we hope that the community will embrace it by contributing additional components.

8 Future Development

8.1 Simulation

Now that the basic SEA tools and infrastructure are available, we can consider prototyping more advanced tools that are built on that infrastructure. One such idea is the concept of observation simulation. We hope to add to the SEA framework additional tools that will allow astronomers to explore the target/instrument/observatory parameters and then “simulate” the quality of data they will attain. For example, a bright target within (or slightly outside) a field of view can have undesirable effects when observing faint sources. These effects not only depend on the location and brightness of the target but on things such as spectral energy distribution (SED), choice of filters, and exposure times. The simulation will convolve the target properties with the instrument setup, known sources of detector defects, quality of available calibrations, and other observing constraints for a given observing mode. Once the basic model is generated, the user will have the ability to interactively manipulate the various observing parameters to visually determine the impact.

For displaying the simulation, we hope to research visual technologies that are just reaching the average desktop such as real-time 3-dimensional rendering. We plan to evaluate the effectiveness of different visual and interactive approaches, and then focus on further developing those that have the biggest impact to our unique astronomical end-user community.

8.2 Natural Language Interface

The Interview user interface prototypes in SEA were built on the assumption that the user is required to describe in great detail every aspect of their proposal. The interview attempts to offer assistance with a few of those details (filter and detector selection, for example), but the real power of this interface is lost because the system requires detail that the interview format cannot provide easily. These details are required, for the most part, so that the observer may optimize their program. However, if we imagine for a moment that the detailed specification of the proposal by the observer is not necessary, we can envision an interview user interface that allows the user to express their proposal in the simplest form possible. Perhaps this UI would mix natural language processing with the question and answer format. For the simplest programs, only a few questions would be necessary. For more complex programs, the tool would know to ask additional questions when more information is required. Other tools might be integrated to confirm steps during the course of the interview (e.g. the VTT could be used to quickly confirm the target field).

This relatively simple proposal would then be fed into the back-end system, as it is now, but that system would include an optimization step in addition to the scheduling and validation steps that now exist. If it worked, this approach would greatly ease the user's proposal definition requirements, while possibly increasing efficiency for the observatory. Increased efficiency might be possible because optimization could occur at the observatory level with exposure granularity, rather than per proposal with visit granularity.

8.3 Expert Systems

Currently an expert system is used in only two small areas in the SEA, in Instrument Configuration (Second Attempt – The Assistant) and in Dithering (Third Attempt – Helpful Observer). We would like to explore a variety of research efforts to determine other useful and appropriate ways to utilize expert systems. We would like to assist the user (if desired) in making decisions based on some basic user provided information. To accomplish this, expert system technology will be used to evaluate the observing model and the options available to the observer. The SEA will provide recommendations not only for the observing strategy, but also for optimizing the observation as well. As a first step, the rulebase will contain rules on how to avoid common observing problems. It can alert the user and suggest best practices to avoid those problems.

We see two primary areas for the expanded use of expert systems in the SEA. We list some possible applications under each area.

- **Enhanced Assistance**

  - Smart Help – an improved context sensitive help system that can do more than simply provide help for a specific field or input box. A "Smart Help" feature would have an underlying rulebase that is evaluating the overall context of the developing proposal and provide more information that is specific and relevant to the context. When the user asks for help, the expert system could determine what help might be most appropriate and make it available. More like discussing an issue with a collaborator or observatory expert

  - Add "Assist" buttons to more of the SEA modules. The assist button is used to request module specific expert system guidance through some portion of the application.

  - Intelligent and unobtrusive monitoring of the proposal so that helpful suggestions can be obtained on demand. The suggestions and the type of help should adapt to the expertise of the user. This feature is particularly helpful to first time or infrequent observers

  - Improved knowledge of each observatory/instrument/detector, their capabilities, the kinds of observations they are "best" at. Make suggestions based on this knowledge. This functionality could be very useful during the initial exploration of an observing program.

- **Functional**

  - An expert system could provide intelligent automatic layout processing in Orbit Planner (Best fit, optimal placement)

  - On demand, the expert system should be able to analyze the current state of the whole proposal, and provide suggestions for optimizing the observing plan. We liken this to final run-through that tax preparation programs like TurboTax or MacInTax provide. The expert system could highlight rough areas or areas that might cause some problems with scheduling, and would ensure that nothing the user specifies can adversely affect the health and safety of the observatory

  - When paired with the Visual Target Tuner (VTT) portion of the SEA, an expert system could provide suggestions on optimal Orientation angles.

  - Determination of effective and efficient mosaic patterns for survey observations of large areas of the sky.

The expert system is there to guide the user and to provide "expert" advice when asked. The user must be able to easily bypass or ignore any and all expert system help. The TurboTax program is a good model. It provides expert help, without forcing you to use it. Tools with expert systems incorporated in them will save both the observer and observatory staff time, because observers can find help as soon as they need it, and user support can be provided with much fewer staff. The result is more cost effective use of everyone's time and effort.

8.4 Adaptation to Production Environments

We are continuing to work closely with STScI to adapt at least two of the SEA's tools, the Visual Target Tuner and the Exposure Time Calculator, to a production release for HST during 2000. This entails reviewing the underlying architecture to ensure that it has robustness for a production rather than prototype environment, reviewing the scientific algorithms closely to ensure accuracy, extending the end-user documentation substantially, and establishing a successful strategy for sharing code between multiple development teams in disparate locations (STScI and Goddard) during parallel development. This effort lays the groundwork for SEA to grow into a multi-observatory tool. With the assistance of the European Southern Observatory (ESO), this technique has been demonstrated for the Very Large Telescope (VLT), allowing the user to compare VLT and HST instruments simultaneously. We have also begun to discuss possible collaborations with other observatories including Gemini, SIRTF, and SOFIA.

8.5 Education and Outreach

One spin-off idea for the SEA, particularly the Visual Target Tuner (VTT) tool, was to adapt the VTT for use in education and/or for amateur astronomers. While attempts to fund this idea were not realized, the idea of adapting the VTT for this purpose still holds promise. The VTT is a powerful, easy way to access otherwise cryptic astronomical data. It would give students access to the same data as professional astronomers. Linked with the public press releases on popular HST images or new discoveries, students could use the VTT to look up the target or image and explore the parameter space with the same tools (or similar) that professional astronomers use. In addition, specific curricula could be designed to work with the HST field of view and apertures, teaching the student how to point the telescope at stars and how to plan a simple observation while simultaneously teaching basic mathematical and physical concepts.

Likewise, the VTT would be a useful tool for the amateur astronomer. Not only would the target visualization features be generally useful, but the VTT architecture supports specifying an arbitrary field of view, including the field of view for an amateur's "backyard" observatory. With some minor effort, the VTT could easily be enhanced to allow the user to input their own telescope's field of view.

9 Conclusions

On a scale of 1 - 5, where 1 was excellent and 5 was poor, the SEA as a whole ranked as 1.8 (i.e., between excellent and above average). All our evaluators (except one!) found using SEA a better and easier experience than RPS2 (the present tool used at STScI to develop proposals). In terms of creating accurate and feasible observations we feel that the SEA provided the proof of concept.

We have looked at a number of possibilities, some like the Visual Target Tuner show great promise, while others such as expert system applications remain intriguing but are still not successful. We strongly believe that visual technologies are a substantial improvement over previous tools, and their impact may well be in areas that are different than we initially expected.

At the start of the SEA project, NGST funded a research effort to evaluate ways to dramatically reduce the time and effort required to support a general observer program. Specifically, the SEA experiment was to determine if new user support tools using advanced technologies could reduce the need for the high level of human user support for routine observations. It is our hope that reduction in support staff will be realized once capabilities prototyped in SEA are fully developed and are available for all observers.

We think the major cost savings will come in the eventual merging of Phase I and Phase II tools, along with support for exploration of an astronomer's initial concept development and research.

We also believe that the trend, while not yet complete, towards collaboration and an evolution to multi-observatory tools is the avenue that promises the greatest cost benefit. There are major cost benefits to observatories if they can support a commonly used tool and adapt it with a small effort, rather than redevelop (and subsequently maintain) a fully custom tool. In addition, there are major productivity gains to both support staff and astronomers if astronomers can use the same basic software suite to develop proposals end-to-end for different observatories. Lastly, if astronomers do not constantly have to learn new preparation tools, their ability to focus on their science will allow substantial improvements in the efficiency and quality of the observations.

Appendix A: Papers Presented by the SEA Group

1. J. Jones, et al, “Lessons Learned from the Scientist’s Expert Assistant Project”, *Proceedings of SPIE Vol. 4010*, 2000.
2. K. Wolf, et al, “Expert System Technology in Observing Tools”, *Proceedings of SPIE Vol. 4010*, 2000.
3. A. Koratkar, et al, “NGST’s Scientist’s Expert Assistant: Evaluation Plans and Results”, *Proceedings of SPIE Vol. 4010*, 2000.
4. A. Koratkar, et al, “Designing the next genera7tion of user support tools: methodology being adopted at Space Telescope Science Institute”, *Proceedings of SPIE Vol. 4010*, 2000.
5. J. Jones, et al, “Next Generation User Support Tools”, *Journal of the Brazilian Society of Mechanical Sciences, Vol. XXI*, pp. 84-89, 1999.
6. A. Koratkar and S. Grosvenor, “The Next Generation User Support Tools”, *Astronomical Society of the Pacific Conference Series, Volume 172*, pp. 57-64, 1998.
7. T. Brooks, et al, “An Expert Assistant System to Support the General Observer Program for NGST”, *Proceedings of SPIE Vol. 3349*, pp. 450-455, 1998.
8. T. Brooks, et al, “Visualization Tools to Support Proposal Submission”, *Proceedings of SPIE Vol. 3349*, pp. 441-449, 1998.
9. T. Brooks, et al, “The Workshop on Observing Tools Summary Report”, *http://aaaprod.gsfc.nasa.gov/workshop/WorkshopReport.htm*, 1998.
10. S. Grosvenor, et al, “Exploring AI Alternatives in Support of the General Observer Program for NGST”, *Proceedings of the International Workshop on Planning and Scheduling for Space Exploration and Science*, 1997.

Appendix B: Deliverables and Presentations

Deliverables

- Requirements Document and Design Document (February 1998)
- Workshop on Observing Tools Summary Report (December 1998)
- Evaluation Plan (May 1999)
- Final Report (March 2000)

- SEA Release 1 (May 1998)
  Initial Framework, basic Exposure Time Calculator and Visual Target Tuner

- SEA Release 2 (September 1998)
  Proposal Browser UI with all modules integrated, Enhanced Visual Target Tuner and Exposure Time Calculator, Target Selector, Proposal Summary Editor, Interview Expert System for ACS filter selection

- SEA Release 3 (April 1999)
  Visit Planner, Exposure Time Calculator spectroscopy support, Proposal Browser UI enhancements, Interview Assistant redesign, substantial Visual Target Tuner enhancements, Apply/Reset option, and InstallAnywhere distribution

- SEA Release 4 (October 1999)
  Orbit Planner, Table Views, Dither Module, Context-sensitive help, XML data files, and additional Exposure Time Calculator and Visual Target Tuner improvements

- SEA Release 4.1 (January 2000)
  Final Release for Evaluation

Presentations

Numerous presentations and demonstrations of the SEA have occurred during the life of the project. The following list includes the highlights of the presentations and posters that were given.

- International Workshop on Planning and Scheduling for Space Exploration and Science, Oxnard, California, October 1997
- SPIE 98, Hawaii, February 1998
- Ad-hoc Science Working Group (ASWG), April 1998
- American Astronomical Society, San Diego, California, June 1998
- NGST Technology Challenge West, Oxnard, California, June 1998
- Goddard Technology Showcase, June 1998
- NGST Workshop, Belgium, June 1998
- Astronomical Data Analysis and Software Systems 98, invited talk, Urbana, Illinois, October 1998
- The Workshop on Observing Tools, College Park, Maryland, October 1998
- American Astronomical Society, Austin, Texas, January 1999
- International Symposium on Spacecraft Ground Control and Data Systems, Iguassu Falls, Brazil, February 1999
- American Astronomical Society, Chicago, Illinois, June 1999
- Astronomical Data Analysis and Software Systems 99, Hawaii, October 1999
- SPIE 2000, March 2000